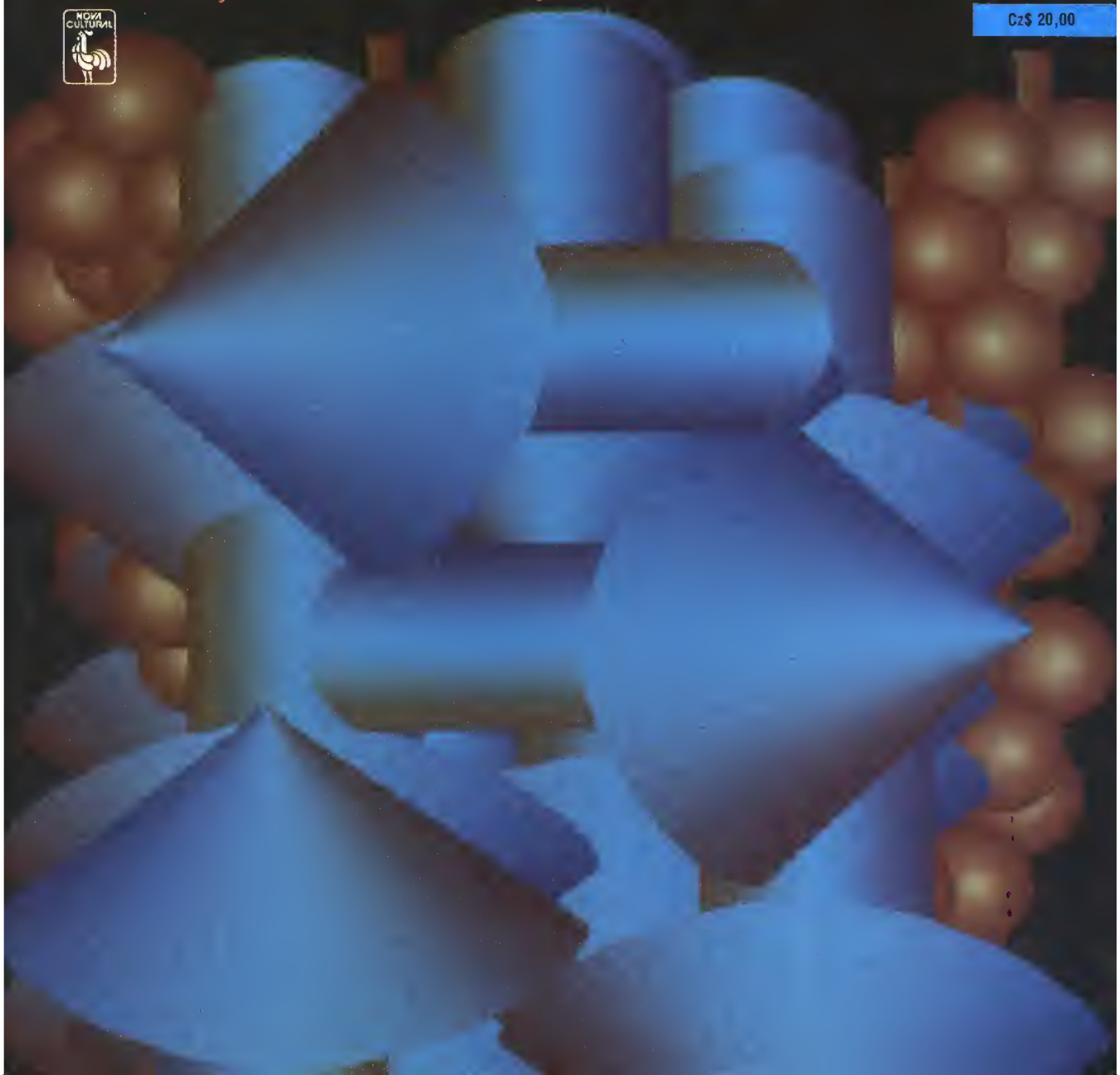
CURSO PRÁTICO 28 DE PROGRAMAÇÃO DE COMPUTADORES

INPUT

PROGRAMAÇÃO BASIC – PROGRAMAÇÃO DE JOGOS – CÓDIGO DE MÁQUINA

NOVA CULTURAL

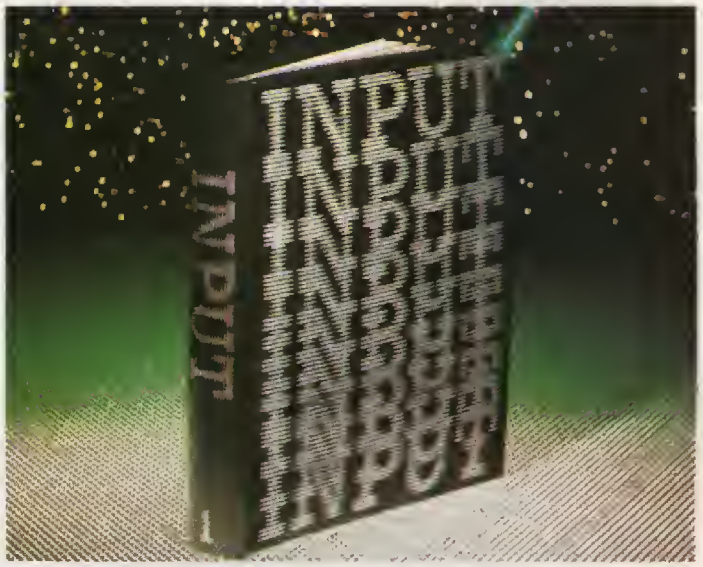
Cz$ 20,00



INPUT

Vol. 2 Nº 28

NESTE NÚMERO



PLANO DA OBRA

"INPUT" é uma obra editada em fascículos semanais, e cada conjunto de 15 fascículos compõe um volume. A capa para encadernação de cada volume estará à venda oportunamente.

COMPLETE SUA COLEÇÃO

Exemplares atrasados, até seis meses após o encerramento da coleção, poderão ser comprados, a preços atualizados, da seguinte forma: **1. Pessoalmente** — por meio de seu jornaleiro ou dirigindo-se ao distribuidor local, cujo endereço poderá ser facilmente conseguido junto a qualquer jornaleiro de sua cidade. Em São Paulo os endereços são: Rua Brigadeiro Tobias, 773 (Centro); Av. Industrial, 117 (Santo André); e, no Rio de Janeiro: Rua da Passagem, 93 (Botafogo). **2. Por carta** — Poderão ser solicitados exemplares atrasados também por carta, que deve ser enviada para DINAP — Distribuidor Nacional de Publicações — Números Atrasados — Estrada Velha de Osasco, 132 (Jardim Tereza) — CEP 06000 — Osasco — São Paulo. **3. Por telex** — Utilize o nº (011) 33670 ABSA. Em **Portugal**, os pedidos devem ser feitos à Distribuidora Jardim de Publicações Ltd. — Qta. Pau Varais, Azinhaga de Fetais — 2685, Camarate — Lisboa; Tel. 257-2542 — Apartado 57 — Telex 43 069 JARLIS P.
Não envie pagamento antecipado. O atendimento será feito pelo reembolso postal e o pagamento, incluindo as despesas postais, deverá ser efetuado ao se retirar a encomenda na Agência do Correio. **Atenção:** Após seis meses do encerramento da coleção, os pedidos serão atendidos, dependendo da disponibilidade de estoque. **Obs.:** Quando pedir livros, mencione sempre o título e/ou o autor da obra, além do número da edição.

COLABORE CONOSCO

Encaminhe seus comentários, críticas, sugestões ou reclamações ao Serviço de Atendimento ao Leitor — Caixa Postal 9442, São Paulo — SP.

Editor
VICTOR CIVITA

REDAÇÃO
Diretora Editorial: Iara Rodrigues

Editor Executivo: Antonio José Filho

Editor Chefe: Paulo de Almeida
Editor de Texto: Cláudio A. V. Cavalcanti
Chefe de Arte: Carlos Luiz Batista
Assistentes de Arte: Ailton Oliveira Lopes, Dilvacy M. Santos, Grace Alonso Arruda, José Maria de Oliveira, Monica Lenardon Corradi
Secretária de Redação/Coordenadora: Stefania Crema
Secretários de Redação: Beatriz Hagström, José Benedito de Oliveira Damião, Maria de Lourdes Carvalho, Marisa Soares de Andrade, Mauro de Queiroz

COLABORADORES
Consultor Editorial Responsável: Dr. Renato M. E. Sabbatini (Diretor do Núcleo de Informática Biomédica da Universidade Estadual de Campinas)
Execução Editorial: DATAQUEST Assessoria em Informática Ltda., Campinas, SP
Tradução: Reinaldo Cúrcio
Tradução, adaptação, programação e redação: Abílio Pedro Neto, Aluísio J. Dornellas de Barros, Marcelo R. Pires Therezo, Raul Neder Porrelli
Coordenação Geral: Rejane Felizatti Sabbatini
Editora de Texto: Ana Lúcia B. de Lucena
Assistente de Arte: Dagmar Bastos Sampaio

COMERCIAL
Diretor Comercial: Roberto Martins Silveira
Gerente Comercial: Flávio Maculan
Gerente de Circulação: Denise Maria Mozol

PRODUÇÃO
Gerente de Produção: João Stungis
Coordenador de Impressão: Atílio Roberto Bonon

Preparador de Texto/Coordenador: Eliel Silveira Cunha
Preparadores de Texto: Alzira Moreira Braz, Ana Maria Dilguerian, Karina Ap. V. Grechi, Levon Yacubian, Luciano Tasca, Maria Teresa Galluzzi, Maria Teresa Martins Lopes, Paulo Felipe Mendrone
Revisor/Coordenador: José Maria de Assis
Revisoras: Conceição Aparecida Gabriel, Isabel Leite de Camargo, Ligia Aparecida Ricetto, Maria de Fátima Cardoso, Nair Lúcia de Brito
Paste-up: Anastase Potaris, Balduino F. Leite, Edson Donato

Edição organizada pela Editora Nova Cultural Ltda.
Av. Brigadeiro Faria Lima, nº 2000 - 3º andar
CEP 01452 - São Paulo - SP - Brasil
(Artigo 15 da Lei 5 988, de 14/12/1973).
Esta obra foi composta na AM Produções Gráficas Ltda. e impressa na Divisão Gráfica da Editora Abril S.A.

CONJUNTOS DE BLOCOS GRÁFICOS (2)

- QUANDO USAR BLOCOS GRÁFICOS
- COMO COMBINAR BLOCOS PARA FORMAR FIGURAS
- CRIAÇÃO DE UM CENÁRIO
- BANCOS DE BLOCOS

Com um conjunto de blocos gráficos, será fácil criar as mais diversas figuras. Você precisará apenas colocar os blocos na posição correta e acrescentar alguns detalhes ao fundo.

Já falamos bastante sobre a criação, proteção, edição e gravação de conjuntos de blocos definidos pelo usuário. Mostramos também ao usuário do Spectrum como exceder o limite de 21 blocos inicialmente imposto. Neste artigo, o segundo de uma série de três, avançamos as explicações sobre o uso de blocos gráficos na criação de figuras na tela de seu microcomputador.

POR QUE USAR BLOCOS GRÁFICOS

Suponhamos que você queira criar uma tela representando uma floresta — para servir de cenário a um jogo de ação, por exemplo. Existem basicamente dois caminhos a seguir: usar os comandos gráficos do BASIC para desenhar cada parte da figura ou, então, combinar blocos gráficos.

Se você escolher a primeira alternativa, terá um trabalho enorme para criar a mata que faz parte do cenário, pois precisará desenhar cada tronco e cada copa de árvore. Além disso, as árvores ficarão muito parecidas umas com as outras.

Se você criar um bloco gráfico, ou vários deles que, combinados, formem uma árvore, poderá colocá-los na tela, repetidas vezes, nas posições que quiser. Você evitará, desse modo, o trabalho de

especificar cada detalhe ao acrescentar mais uma árvore na tela.

Existem ainda outras vantagens no uso de UDG. Uma delas é a economia de tempo. Os desenhos compostos por blocos gráficos, não importa se mais ou menos complicados, são traçados na tela numa velocidade maior que os produzidos com comandos gráficos do BASIC. Assim, se compusermos uma figura com blocos gráficos, teremos que esperar bem menos tempo por sua aparição na tela.

LIBERDADE DE ESCOLHA

Outra vantagem é a facilidade de variar o tamanho e a forma da figura. No desenho da floresta, por exemplo, poderíamos mudar o tamanho de uma árvore aumentando ou diminuindo o número de "blocos de tronco" e, ainda, alterar sua copa usando uma combinação diferente de "blocos de folhagem". Os comandos gráficos do BASIC podem fazer a mesma coisa, mas, com os blocos, a tarefa é bem mais fácil.

Uma vez criados, os blocos gráficos permanecem na memória até serem modificados — ou o micro ser desligado. No caso do Spectrum, se dispusermos de vários bancos, precisaremos "ligar" o banco que estiver sendo usado naquele momento. Mas isto é só uma questão de modificar o apontador de UDG.

Poderemos, assim, usar os blocos que criamos quantas vezes quisermos dentro de um programa: o único limite é a memória. Voltando ao nosso exemplo, será tão fácil criar uma floresta quanto uma única árvore, utilizando UDG.

É claro que o emprego de blocos gráficos também apresenta desvantagens. Para começar, temos que definir todos os blocos, recorrendo ao gerador ou digitando várias linhas **DATA**. Vale lembrar, porém, que, usando comandos gráficos, teremos ainda mais trabalho.

Outra desvantagem é a ocupação de parte da preciosa memória do micro para guardar os blocos e, pior, a menos que sejamos cuidadosos, acabamos fazendo-o duas vezes. No próximo artigo, veremos como contornar o problema.

QUANDO VALE A PENA USAR UDG

Por todas estas razões, é mais vantajoso usar blocos gráficos em certas figuras do que em outras.

Como regra geral, se o cenário inclui uma série de objetos semelhantes, desenhados várias vezes, ou se algum tipo de figura aparece repetidamente no decorrer do programa, o uso de UDG vai nos poupar tempo e trabalho.

Se, por outro lado, queremos um desenho muito detalhado, que usaremos apenas uma vez durante o programa, pode ser melhor ficarmos com os comandos gráficos do BASIC.

Uma parede de tijolos, por exemplo, poderá ser desenhada com muito mais facilidade se utilizarmos um ou vários blocos repetidas vezes. Uma alternativa seria traçar diversas linhas sobre uma região colorida para imitar os espaços entre os tijolos.

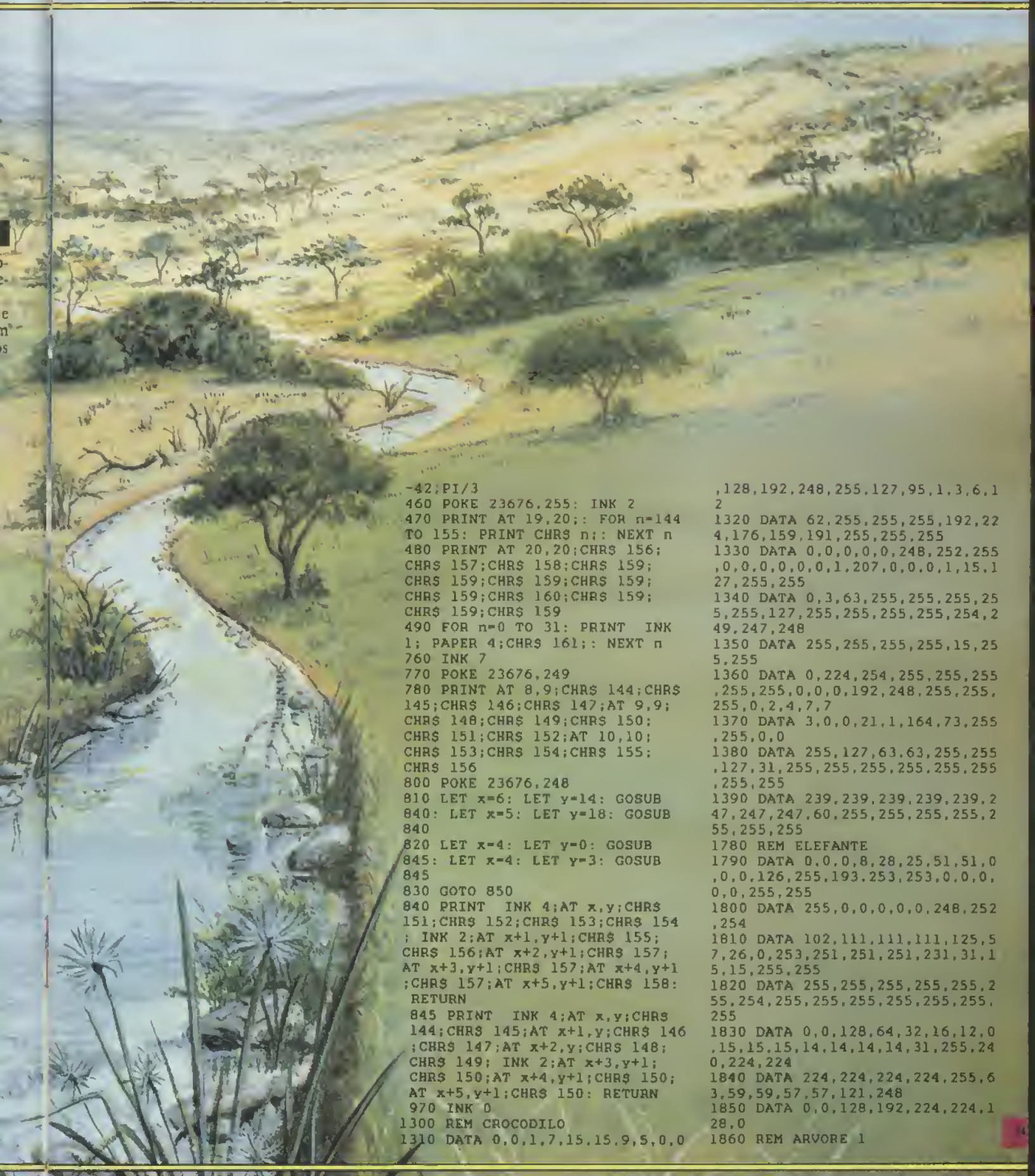
No nosso cenário de floresta, já vimos que podemos fazer uma boa economia de tempo criando UDG para as árvores que nele aparecem. Os animais também podem ser desenhados com blocos gráficos, principalmente se formos utilizá-los novamente no decorrer do programa, ou se quisermos animá-los ou desenhá-los várias vezes.

UMA FLORESTA NA TELA

Apresentaremos a seguir alguns programas que criam os caracteres para desenhar o cenário já mencionado.

As linhas **DATA** para o Spectrum e o MSX são as mesmas. Elas começam na linha 1300 e estão listadas logo após o programa do MSX.

```
 10 CLEAR 63500
110 POKE 23676,255
120 FOR n=USR "a" TO USR "r"+7
: READ a: POKE n,a: NEXT n
260 POKE 23676,249
270 FOR n=USR "a" TO USR "m"+7
: READ a: POKE n,a: NEXT n
290 POKE 23676,248
300 FOR n=USR "a" TO USR "o"+7
: READ a: POKE n,a: NEXT n
410 BORDER 1: PAPER 8: CLS
420 FOR n=1 TO 8: PRINT  PAPER
5;" ";TAB 31;" ": NEXT n
430 FOR n=1 TO 14: PRINT
PAPER 4;" ";TAB 31;" ": NEXT n
440 PLOT 0,110: DRAW 142,-100,
-PI/3: PLOT 160,110: DRAW -60,
```

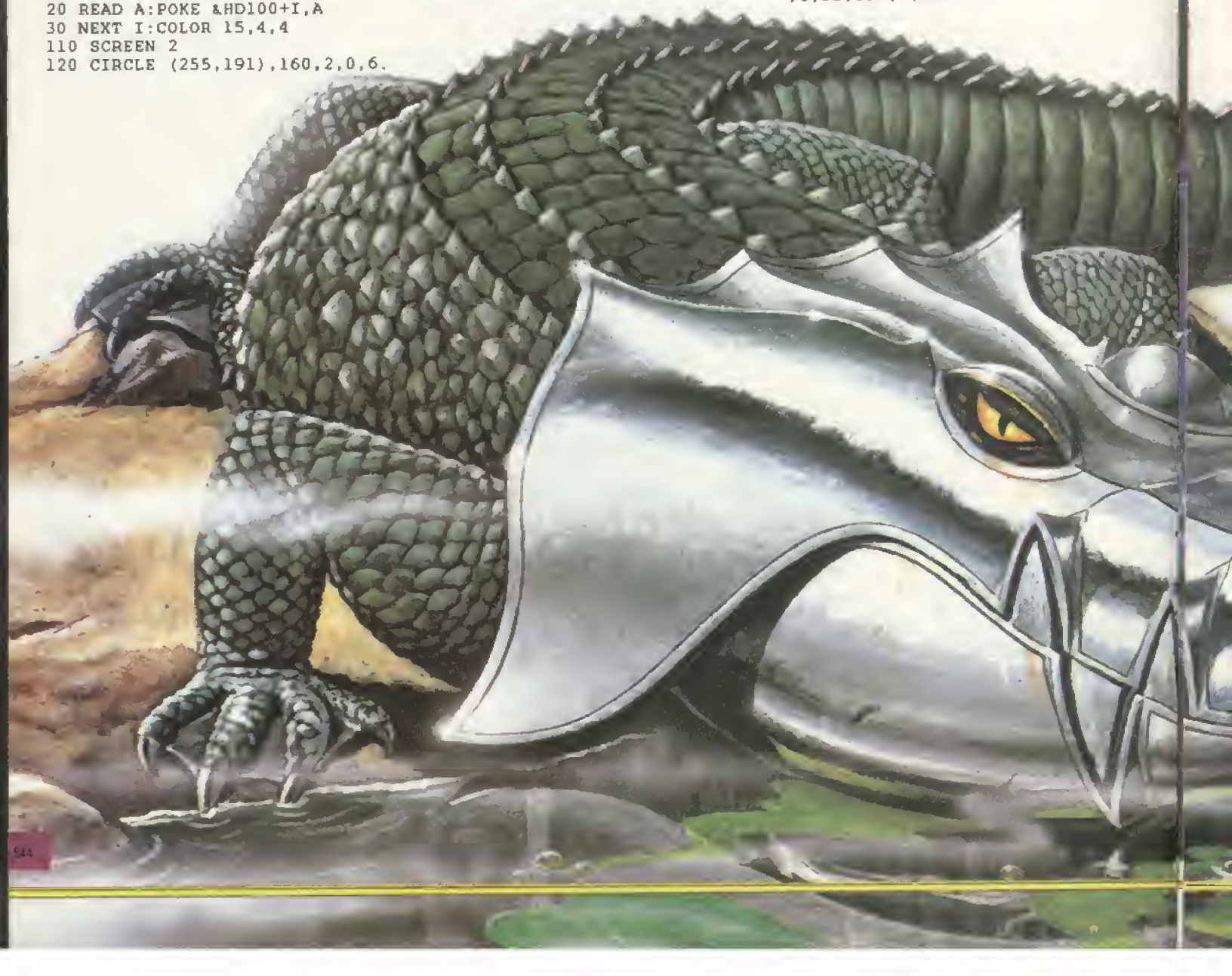


```
-42:PI/3
460 POKE 23676,255: INK 2
470 PRINT AT 19,20;: FOR n=144
TO 155: PRINT CHR$ n;: NEXT n
480 PRINT AT 20,20;CHR$ 156;
CHR$ 157;CHR$ 158;CHR$ 159;
CHR$ 159;CHR$ 159;CHR$ 159;
CHR$ 159;CHR$ 160;CHR$ 159;
CHR$ 159;CHR$ 159
490 FOR n=0 TO 31: PRINT  INK
1; PAPER 4;CHR$ 161;: NEXT n
760 INK 7
770 POKE 23676,249
780 PRINT AT 8,9;CHR$ 144;CHR$
145;CHR$ 146;CHR$ 147;AT 9,9;
CHR$ 148;CHR$ 149;CHR$ 150;
CHR$ 151;CHR$ 152;AT 10,10;
CHR$ 153;CHR$ 154;CHR$ 155;
CHR$ 156
800 POKE 23676,248
810 LET x=6: LET y=14: GOSUB
840: LET x=5: LET y=18: GOSUB
840
820 LET x=4: LET y=0: GOSUB
845: LET x=4: LET y=3: GOSUB
845
830 GOTO 850
840 PRINT  INK 4;AT x,y;CHR$
151;CHR$ 152;CHR$ 153;CHR$ 154
; INK 2;AT x+1,y+1;CHR$ 155;
CHR$ 156;AT x+2,y+1;CHR$ 157;
AT x+3,y+1;CHR$ 157;AT x+4,y+1
;CHR$ 157;AT x+5,y+1;CHR$ 158:
 RETURN
 845 PRINT  INK 4;AT x,y;CHR$
 144;CHR$ 145;AT x+1,y;CHR$ 146
 ;CHR$ 147;AT x+2,y;CHR$ 148;
 CHR$ 149; INK 2;AT x+3,y+1;
 CHR$ 150;AT x+4,y+1;CHR$ 150;
 AT x+5,y+1;CHR$ 150: RETURN
 970 INK 0
1300 REM CROCODILO
1310 DATA 0,0,1,7,15,15,9,5,0,0
,128,192,248,255,127,95,1,3,6,1
2
1320 DATA 62,255,255,255,192,22
4,176,159,191,255,255,255
1330 DATA 0,0,0,0,0,248,252,255
,0,0,0,0,0,0,1,207,0,0,0,1,15,1
27,255,255
1340 DATA 0,3,63,255,255,255,25
5,255,127,255,255,255,255,254,2
49,247,248
1350 DATA 255,255,255,255,15,25
5,255
1360 DATA 0,224,254,255,255,255
,255,255,0,0,0,192,248,255,255,
255,0,2,4,7,7
1370 DATA 3,0,0,21,1,164,73,255
,255,0,0
1380 DATA 255,127,63,63,255,255
,127,31,255,255,255,255,255,255
,255,255
1390 DATA 239,239,239,239,239,2
47,247,247,60,255,255,255,255,2
55,255,255
1780 REM ELEFANTE
1790 DATA 0,0,0,8,28,25,51,51,0
,0,0,126,255,193,253,253,0,0,0,
0,0,255,255
1800 DATA 255,0,0,0,0,0,248,252
,254
1810 DATA 102,111,111,111,125,5
7,26,0,253,251,251,251,231,31,1
5,15,255,255
1820 DATA 255,255,255,255,255,2
55,254,255,255,255,255,255,255,
255
1830 DATA 0,0,128,64,32,16,12,0
,15,15,15,14,14,14,14,31,255,24
0,224,224
1840 DATA 224,224,224,224,255,6
3,59,59,57,57,121,248
1850 DATA 0,0,128,192,224,224,1
28,0
1860 REM ARVORE 1
```

```
1870 DATA 0,0,0,0,1,1,3,7,0,0,0
,0,224,240,248,248,15,63,63,63.
31,15,3,3
1880 DATA 252,254,254,254,254,2
54,252,252
1890 DATA 3,3,3,3,3,1,0,0,248,2
48,248,248,248,248,240,96,96,96
,96,96,96,96
1900 DATA 96,96
1910 REM ARVORE 2
1920 DATA 0,3,15,31,127,127,63,
1,7,15,255,255,255,255,255,255.
15,63,255,255
1930 DATA 255,255,255,255,0,128
,248,248,248,248,240,224
1940 DATA 255,227,96,48,24,25,1
3,15,252,240,96,96,192,192,128.
128,7,7,7,7,7
1950 DATA 7,7,7,7,7,7,15,15,15,
31,63
```

```
5 CLEAR 200,&HC999
10 FOR I=0 TO 367
20 READ A:POKE &HD100+I,A
30 NEXT I:COLOR 15,4,4
110 SCREEN 2
120 CIRCLE (255,191),160,2,0,6.
28,.6:PAINT(230,191),2
130 CIRCLE(0,191),143,2,0,6.28,
.4:PAINT(10,191),2
140 CIRCLE(0,191),143,12,0,6.28
,.4
200 FOR I=1 TO 51
210 READ A,B,C:FOR J=0 TO 7
220 VPOKE BASE(12)+A*8+J,PEEK(&
HD100+B*8+J)
230 VPOKE BASE(11)+A*8+J,C
```

```
240 NEXT J,I
250 FOR K=1 TO 6:READ D(K):NEXT
 K
260 FOR I=1 TO 8:READ A(I),B(I)
,C(I):NEXT I:FOR K=1 TO 6:FOR I
=1 TO 8:FOR J=0 TO 7
270 VPOKE BASE(12)+(A(I)+D(K))*
8+J,PEEK(&HD100+B(I)*8+J)
280 VPOKE BASE(11)+(A(I)+D(K))*
8+J,C(I)
290 NEXT J,I,K
300 FOR K=1 TO 8:READ D(K):NEXT
 K
310 FOR I=1 TO 9:READ A(I),B(I)
,C(I):NEXT I:FOR K=1 TO 8:FOR I
=1 TO 9:FOR J=0 TO 7
320 VPOKE BASE(12)+(A(I)+D(K))*
8+J,PEEK(&HD100+B(I)*8+J)
330 VPOKE BASE(11)+(A(I)+D(K))*
8+J,C(I)
340 NEXT J,I,K
400 GOTO 400
3000 DATA 692,0,98,693,1,98,694
,2,98,695,3,98
3010 DATA 696,4,98,697,5,98,698
,6,98,699,7,98
```



```
3020 DATA 700,8,98,701,9,98,702
,10,98,703,11,98
3030 DATA 724,12,98,725,13,98,7
26,14,98,727,15,98
3040 DATA 728,15,98,729,15,98,7
30,15,98,731,15,98
3050 DATA 732,16,98,733,15,98,7
34,15,98,735,15,98
3060 DATA 754,17,78,755,17,78,7
56,17,78,757,17,78,758,17,78,75
9,17,78
3070 DATA 760,17,78,761,17,78,7
62,17,78,763,17,78
3080 DATA 764,17,78,765,17,78,7
66,17,78,767,17,78
3090 DATA 500,18,226,501,19,226
,502,20,226,503,21,226
3100 DATA 532,22,226,533,23,226
,534,24,226,535,25,226
3110 DATA 536,26,226,565,27,226
,566,28,226,567,29,226,568,30,2
26
3115 DATA -32,-4,36,75,80,147
3120 DATA 342,31,36,343,32,36,3
74,33,36,375,34,36
3130 DATA 406,35,36,407,36,36,4
39,37,96,471,37,96
3135 DATA -44,-20,-15,60,84,96,
157,163
3140 DATA 327,38,36,328,39,36,3
29,40,36,330,41,36
3150 DATA 360,42,96,361,43,96,3
92,44,96,424,44,96
3160 DATA 456,45,96
```

S

```
1300 REM CROCODILO
1310 DATA 0,0,1,7,15,15,9,5,0,0
,128,192,248,255,127,95,1,3,6,1
2
1320 DATA 62,255,255,255,192,22
4,176,159,191,255,255,255
1330 DATA 0,0,0,0,0,248,252,255
,0,0,0,0,0,0,1,207,0,0,0,1,15,1
27,255,255
1340 DATA 0,3,63,255,255,255,25
5,255,127,255,255,255,255,254,2
49,247,248
1350 DATA 255,255,255,255,15,25
5,255
1360 DATA 0,224,254,255,255,255
,255,255,0,0,0,192,248,255,255,
255,0,2,4,7,7
1370 DATA 3,0,0,21,1,164,73,255
,255,0,0
1380 DATA 255,127,63,63,255,255
,127,31,255,255,255,255,255,255
,255,255
1390 DATA 239,239,239,239,239,2
47,247,247,60,255,255,255,255,2
55,255,255
1780 DATA REM ELEFANTE
1790 DATA 0,0,0,8,28,25,51,51,0
,0,0,126,255,193,253,253,0,0,0,
0,0,255,255
1800 DATA 255,0,0,0,0,0,0,248,252
,254
1810 DATA 102,111,111,111,125,5
7,26,0,253,251,251,251,231,31,1
5,15,255,255
1820 DATA 255,255,255,255,255,2
55,254,255,255,255,255,255,255,
255
1830 DATA 0,0,128,64,32,16,12,0
,15,15,15,14,14,14,14,31,255,24
0,224,224
1840 DATA 224,224,224,224,255,6
3,59,59,57,57,121,248
1850 DATA 0,0,128,192,224,224,1
28,0
1860 REM ARVORE 1
1870 DATA 0,0,0,0,1,1,3,7,0,0,0
,0,224,240,248,248,15,63,63,63,
31,15,3,3
1880 DATA 252,254,254,254,254,2
54,252,252
1890 DATA 3,3,3,3,3,1,0,0,248,2
48,248,248,248,240,96,96,96
,96,96,96,96
1900 DATA 96,96
1910 REM ARVORE 2
1920 DATA 0,3,15,31,127,127,63,
1,7,15,255,255,255,255,255,255,
15,63,255,255
1930 DATA 255,255,255,255,0,128
,248,248,248,248,240,224
1940 DATA 255,227,96,48,24,25,1
3,15,252,240,96,96,192,192,128,
128,7,7,7,7,7
1950 DATA 7,7,7,7,7,7,15,15,15,
31,63
```

T

```
10 CLEAR 1000:CLS
20 DIM C(59),E(17),T1(1),T2(7),
F1(7),F2(7)
30 PMODE 3,1:PCLS
40 W$="L6UL6D":W$=W$+W$+W$+W$
50 DRAW"BM1,4C4RUR2UR2DR2DR4DR8
UR4UR2UR2UR2UR4DR2DR2DR4D2R6DR2
DR4DR4UR4UR4UR4UR4UR4UR4UR10
DR6DR4DR2DR2DR2D15"+W$+W$+"L6UL
6DL4BU14DR11FRFR7FR3FR9FGL7GL21
HL2D3FR21FR3F"
60 PAINT(50,10),4
70 DRAW"BM98,5C1L8GLGLGD5BFD2RF
R3FBM32,2GFREHLBM1,8C2FRERFBR4U
BM+4,2;RBR3RBM+3,1;RBM+3,1;RBR3
RBL8BDLBL5NEBL6NEBL4EBL5E"
80 GET (0,0)-(112,20),C,G
230 PCLS
240 DRAW"BM4,0C2DG2DG2D3R2DE2UE
2UE43FRFR11F3RFRFR3DBL8NU3D4F2D
GH2UHL3D5LURU3HL6D5L2BU6L3D6NL2
U6LH2LNGUHL2"
250 PAINT(20,7),2
260 DRAW"BM12,3C3R2D4G2BM8,4R"
270 GET(0,0)-(37,17),E,G
280 PCLS 4
290 DRAW"BM7,19C1H3U5H5UE6R2F3D
F2D2G3D3G2D2":PAINT(7,7),1
300 DRAW"BM20,5E2R2E2RFR4E2F2R2
E2R5FRFDG3LGLGLGL5HL5H2L4":PAIN
T(34,5),1
310 GET(0,0)-(13,19),F1,G:GET(2
0,0)-(49,9),F2,G
```

```
320 PCLS:DRAW"BM0,0C4D20BE20F3D
FD15G2NL2R8HL4EU13E4U2"
330 GET(0,0)-(1,20),T1,G:GET(20
,0)-(31,22),T2,G
340 PCLS3:SCREEN 1,0
350 CIRCLE(255,191),160,1,.6:PA
INT(230,180),1
360 CIRCLE(0,191),140,2,.35,.75
,1:PAINT(10,180),1,2
380 PUT(206,100)-(243,117),E,PS
ET
390 FOR K=1 TO 10: READ X,Y:PUT
(X,Y)-(X+13,Y+19),F1,AND:PUT(X+
6,Y+20)-(X+7,Y+40),T1,OR:NEXT
400 FOR K=1 TO 10:READ X,Y:PUT(
X,Y)-(X+29,Y+9),F2,AND:PUT(X+8,
Y+9)-(X+19,Y+31),T2,OR:NEXT
410 COLOR 2:LINE (138,187)-(255
,187),PSET:PAINT(255,191),2:PAI
NT(255,191),3,1
420 PUT(143,166)-(255,186),C,PS
ET
450 DATA 16,110,24,113,34,108,4
8,110,56,108,190,80,198,82,212,
84,210,79,240,70
460 DATA 2,120,18,122,28,116,46
,118,60,124,160,90,174,95,190,9
0,214,86,226,90
470 GOTO 470
```

Executando estes programas, veremos o tipo de figura que podemos criar usando UDG. Não se preocupe se o alto da tela parece vazio no momento, pois o cenário ainda não está completo.

A água sob o crocodilo é uma boa demonstração da versatilidade dos blocos gráficos: ela é composta pela repetição de um só bloco. (Isto não vale para o TRS-Color, onde a matriz do crocodilo inclui a água.)

O cenário não é formado apenas por UDG: empregamos também comandos gráficos para desenhar as colinas. Embora com freqüência seja melhor usar blocos gráficos em vez de comandos gráficos, podemos obter ótimos resultados combinando as duas técnicas. O próximo artigo completará o desenho; portanto, grave o programa em fita.

COMO FUNCIONA

Se você não entendeu como o programa define e guarda os blocos, consulte o último artigo desta série.

O programa pode ser dividido em duas partes: uma que define os blocos e outra que os coloca na tela.

O Spectrum utiliza o comando **POKE** para colocar os valores das linhas **DATA** na tela; o MSX, por sua vez, usa **VPOKE**. No programa do TRS-Color não há linhas **DATA**: desenhamos as figuras com **DRAW** e, em seguida, guardamos os padrões em matrizes com **GET**.

Criados os blocos, o programa prossegue colocando-os nos locais adequados.

S

O Spectrum usa uma série de comandos **PRINT AT** para desenhar cada segmento de animal ou árvore. Ele também emprega comandos locais de cor para colorir as figuras.

Um comando local de cor é aquele que se aplica somente à linha onde ocorre. Em geral, fazemos com que o pano de fundo — **PAPER** — seja 8, que é transparente. Isso evita que os blocos impressos na tela o danifiquem.

UM PANO DE FUNDO MULTICOLORIDO

O pano de fundo é definido pelas linhas 410 a 440. A primeira delas estabelece a cor, e as duas seguintes produzem o céu azul-claro e o chão verde. Para conseguir as diferentes cores de fundo, imprimem-se espaços com as cores escolhidas, por meio de dois laços **FOR...NEXT**.

Como existem várias árvores no cenário, elas são desenhadas com subrotinas (linhas 810 e 820). As variáveis x e y correspondem às coordenadas de comandos **PRINT AT**. Podemos facilmente acrescentar novas árvores ao desenho, colocando mais coordenadas (veja, no artigo da página 501, como escolher as coordenadas adequadas) e comandos **GOSUB** nessas duas linhas.

MSX

O programa do MSX usa o comando **VPOKE** para colocar os blocos na tela gráfica. Comandos gráficos do BASIC também são empregados para desenhar o fundo da figura.

Inicialmente, a linha 5 protege o topo da memória para que as linhas 10 a 30 coloquem ali os padrões dos blocos. A linha 30 também seleciona as cores da tela; a 110 seleciona a tela. As linhas 120 a 140 desenham as duas colinas que aparecem no fundo.

Em seguida, usando **VPOKE**, os padrões são colocados na tela gráfica. A posição que o bloco deve ocupar na tela — A —, a posição do bloco no banco de blocos — B — e a cor do bloco — C — são obtidos com **READ** nas linhas **DATA** que começam em 3000.

Já que existem dois tipos de árvore, elas são desenhadas por dois laços **FOR...NEXT**. Precisamos, no entanto, de um grande número delas. Para fazê-las, usamos uma série de valores que se somam às posições em que cada árvore é desenhada. Esses valores são lidos nas linhas **DATA** 3115 e 3135, mudando a posição de cada árvore sempre que o laço é executado.

As cores não foram colocadas no banco, e sim nas linhas **DATA** a partir de 3000, juntamente com as posições dos blocos. Isso foi possível porque, em nosso exemplo, cada bloco tinha apenas duas cores. Quando o colorido dos blocos for mais complicado, convém usar o banco.

T

O programa do TRS-Color começa reservando espaço para os cordões que usa e **DIM**ensionando as matrizes que guardam os blocos gráficos. Com exceção das árvores, temos um bloco gráfico para cada figura, já que os blocos podem ter qualquer tamanho. As árvores requerem dois blocos cada, pois têm uma metade vermelha e outra verde.

OS ANIMAIS

O programa prossegue desenhando um animal de cada vez e guardando cada um deles na matriz apropriada, com o comando **GET**. A linha 80 guarda o crocodilo, a linha 270 o elefante, a linha 310 cuida das copas das árvores e a linha 330 dos troncos.

A segunda parte do programa coloca cada parte do desenho em seu lugar. As primeiras linhas limpam e selecionam a tela apropriada, desenhando também as colinas que fazem parte do cenário. As cores das colinas são determinadas pelo comando **PAINT**.

Os dois grupos de árvores, cada um em uma colina, são desenhados pelo laço **FOR...NEXT** das linhas 390 e 400, usando as linhas **DATA** 450 e 460 para determinar a posição.

As linhas 380 e 420 colocam o elefante e o crocodilo na tela, respectivamente. A última linha ativa do programa (se ignorarmos as linhas **DATA**) é a 470. Ela impede que o programa termine, permitindo a visualização da tela que, de outra forma, seria apagada. Use <**BREAK**> para parar o programa.

UMA MANADA DE ELEFANTES

Podemos substituir o elefante por uma manada. O método é o mesmo utilizado para as árvores: um laço **FOR...NEXT** ou vários comandos **GOSUB**.

Já vimos como usar blocos gráficos para escrever na tela de alta resolução.

Agora vamos examinar como carregar bytes na área reservada ao banco de blocos — segunda página de vídeo — por meio do monitor.

O monitor permite que coloquemos valores diretamente na memória do micro. Estes números serão, em nosso caso, os bytes correspondentes aos blocos gráficos necessários para o cenário.

Depois de colocar todos estes bytes na memória, carregue e execute o gerador de blocos, para poder ver, editar e entender como foram criadas as figuras. No próximo artigo apresentaremos o programa BASIC que desenha a floresta usando o banco de blocos que estamos criando hoje.

Para carregar o banco de blocos, ative o monitor com:

```
CALL - 151
```

ou **LM** no TK-2000.

Agora, copie a listagem. Para alterar o conteúdo de uma posição de memória, digite o endereço pretendido, seguido de dois pontos e dos valores desejados. Em nosso exemplo, modificamos oito posições de cada vez. Lembre-se de que todos os números precisam estar na forma hexadecimal. No TK-2000, devem-se usar outros endereços. Em vez de começar no endereço 4000, em hexa, use A000, também em hexa.

Note que nem todas as posições da área do banco precisam ser modificadas, havendo várias lacunas na listagem.

```
4008: 00 00 50 54 54 54 44 44
4010: 00 00 00 02 0A 2A 2A 08
4018: 00 40 40 50 14 55 55 55
4020: 02 0A 0A 2A 28 2A 2A 2A
4028: 00 00 00 00 00 05 15 55
4030: 00 00 00 00 00 00 28 2A
4038: 00 00 00 40 50 54 55 55
4040: 00 20 2A 2A 2A 2A 2A 2A
4048: 54 55 55 55 55 55 05 51
4050: 2A 2A 2A 2A 2A 00 2A 2A
4058: 55 55 55 55 55 55 55 55
4060: 00 00 0A 2A 2A 2A 2A 2A
4068: 00 00 00 01 15 55 55 55
4070: 00 00 00 00 00 02 0A 2A
4078: C0 D0 D4 D4 C4 C4 C4 80
4080: AA 8A 82 80 80 80 80 80
4088: 80 80 80 80 C0 C0 D0 D0
4090: AA AA AA AA 8A 8A 82 82
4098: 81 81 81 81 80 80 80 80
40A8: D0 C0 80 80 80 80 80 80
40B0: 8A 82 80 80 80 80 80 80
40C8: C0 C0 90 90 84 84 81 81
40F0: 00 00 20 28 28 28 2A 2A
40F8: 40 54 55 55 55 55 55 55
4100: 2A 2A 2A 2A 2A 2A 2A 2A
4108: 55 55 55 55 55 55 55 55
4110: 2A 2A 2A 2A 2A 2A 2A 2A
4118: 55 55 55 55 55 55 55 55
4120: 2A 2A 2A 2A 2A 2A 2A 2A
4128: 01 15 55 55 55 55 55 55
4130: 00 00 02 0A 0A 0A 2A 2A
4148: 10 14 54 54 54 00 00 00
4150: 00 02 22 2A 2A 00 00 00
4158: 40 44 15 55 55 00 00 00
4160: 2A 2A 2A 2A 2A 28 20 20
4168: 55 55 55 55 55 55 55 55
4170: 2A 2A 2A 2A 2A 2A 2A 2A
4178: 55 55 55 55 55 55 55 55
4180: 2A 2A 2A 2A 2A 2A 2A 2A
4188: 51 51 51 51 51 45 45 55
4190: 2A 2A 2A 2A 2A 2A 2A 2A
4198: 55 55 55 55 55 55 55 55
41A0: 2A 2A 2A 2A 2A 2A 2A 2A
41A8: 55 55 55 55 55 55 55 55
41B0: 2A 2A 2A 2A 2A 2A 2A 2A
41C8: D0 D0 D0 D0 D0 D4 D4 D4
41D0: 80 80 80 80 A0 A8 AA AA
41D8: 80 80 80 85 95 95 D5 D0
41E0: 80 80 80 80 80 80 80 82
41F8: C0 D0 D0 D0 D0 D0 D0 D0
4200: 82 AA 82 82 82 80 80 80
4230: 2A 2A 2A 2A 2A 2A 2A 2A
4238: 55 55 55 55 55 55 55 55
4240: 2A 2A 2A 2A 2A 2A 2A 2A
4248: 55 55 55 55 55 55 55 55
4250: 2A 2A 2A 2A 2A 2A 2A 2A
4258: 55 55 55 55 55 55 55 55
4260: 2A 2A 2A 2A 2A 2A 2A 2A
4268: 55 55 55 55 55 55 55 55
4270: 2A 2A 2A 2A 2A 2A 2A 2A
4280: AA D5 D5 D5 D5 D5 D5 D5
4288: AA AA AA AA AA AA AA AA
4290: AA D5 D5 D5 D5 D5 D5 D5
4298: A5 AA AA AA AA AA AA AA
42A0: AA D5 D5 D5 D5 D5 D5 D5
42A8: A5 AA AA AA AA AA AA AA
42B0: AA D5 D5 D5 D5 D5 D5 D5
42B8: A5 AA AA AA AA AA AA AA
42C0: AA D5 D5 D5 D5 D5 D5 D5
42C8: A5 AA AA AA AA AA AA AA
42D0: AA D5 D5 D5 D5 D5 D5 D5
42D8: A5 AA AA AA AA AA AA AA
42E0: AA D5 D5 D5 D5 D5 D5 D5
42E8: A5 AA AA AA AA AA AA AA
42F0: AA D5 D5 D5 D5 D5 D5 D5
4308: C0 C0 C0 C0 C0 C0 C0 C0
4310: AA AA 8A 8A 8A 8A 8A 8A
4318: D0 D1 D1 D1 95 84 80 80
4320: 82 82 82 80 80 80 80 80
4338: D0 D0 D0 D4 D4 D5 D5 D5
4340: 80 80 80 80 80 80 80 80
4370: 2A 2A 28 28 28 20 00 00
4378: 55 55 55 55 55 55 54 40
4380: 2A 2A 2A 2A 2A 2A 2A 2A
4388: 90 90 D0 D0 C0 C0 55 55
4390: 80 80 80 80 A2 A2 A2 AA
4398: 84 84 85 85 81 81 55 55
43A0: 2A 2A 2A 2A 2A 2A 2A 2A
43A8: 55 55 55 55 55 55 15 01
43B0: 2A 2A 0A 0A 0A 02 00 00
4448: C0 C0 C0 C0 C0 C0 80 80
4450: 8A 8A 8A 8A 8A 8A AA AA
4470: A0 A0 A8 A8 AA AA AA AA
4478: 95 85 85 81 81 80 80 80
44D0: AA AA AA AA AA AA AA AA
4500: 00 00 00 00 00 00 40 50
4508: 00 00 00 00 00 00 7E 3A
4510: 00 00 00 00 00 00 01 15
4518: 00 00 00 00 00 60 68 7A
4520: 00 00 00 00 55 55 55 55
4528: 00 00 00 00 2A 2A 2E 2E
4530: 00 00 00 00 01 05 15 7D
4590: AA AA A8 A8 A8 A8 A0 A0
4598: 80 81 81 81 81 85 85 95
45B0: AA AA AA AA AA A8 A8 A8
45B8: D0 D4 D4 D4 D0 C1 C1 D1
45C0: A8 AA AA 8A AA AA A8 A8
45C8: B5 95 95 95 95 91 95 95
4610: AA AA AA AA AA AA AA AA
4640: 50 50 54 54 54 FD 7F 5D
4648: 3A 3A 2A 22 00 00 20 2A
4650: 55 45 45 45 45 14 14 50
4658: 2A 2A 6A 7E 6A 2A 2A 2A
4660: 55 55 05 01 01 01 05 55
4668: 2E 3F 2A 20 00 00 20 6A
4670: FD 5D 55 05 50 55 5D 57
4678: 00 02 00 2A 2A 2E 2E 2E
4680: 00 00 00 55 7F 5D 5D 5D
4688: 00 00 00 02 2A 3A 3A 3A
4690: 00 00 50 14 15 55 55 55
4698: 20 2A 2A 2A 2A 3E 0E 0F
46A0: 01 05 05 05 04 04 04 00
46D8: D5 D5 D4 D4 D4 D5 D5 D5
46E0: 80 AA AA AA AA AA AA AA
46E8: 80 D5 D5 D5 D5 D5 D5 D5
46F0: A8 AA AA AA AA AA AA AA
46F8: D5 D5 D5 D5 D5 D5 D5 D5
4700: AA 82 AA AA 82 80 80 80
4708: 95 95 D5 D5 D4 D4 94 80
4710: 80 A8 80 80 88 88 A8 AA
4718: 80 81 81 81 81 81 81 81
4750: AA AA AA AA AA AA AA AA
4780: 54 50 00 00 00 00 00 00
4788: 3A 0A 00 00 00 00 00 00
4790: 50 40 40 00 00 00 00 00
4798: 68 6A 62 2A 0A 2A 28 00
47A0: 5F 57 55 D5 55 00 55 55
47A8: 2B 2B 2B 2F 2A 00 2A 2A
47B0: 57 57 57 15 11 04 05 01
47B8: 3F 2B 02 00 00 00 00 00
47C0: 55 55 00 00 00 00 00 00
47C8: 7E 2A 00 00 00 00 00 00
47D0: 55 55 55 54 50 40 40 00
47D8: 0B 03 2A 03 02 02 0A 2A
47E0: 00 10 05 10 00 00 00 00
4810: 80 80 80 A0 A0 A0 A0 A0
4818: D5 D5 D5 D5 D5 D5 D5 D5
4820: AA AA AA AA AA AA AA AA
4828: D5 D5 D5 D5 D5 D5 D5 D5
4830: AA AA AA AA AA AA AA AA
4838: D5 D5 D5 D5 D5 D5 D5 D5
4840: 82 8A 8A 8A 8A 8A 82 82
4848: C0 C0 C0 D0 D0 D0 94 95
4850: AA 8A 8A 82 82 80 80 80
4888: 00 00 00 00 C0 C0 C0 D0
4890: AA AA AA AA AA AA AA AA
4898: 00 00 00 00 81 81 81 85
4948: 80 80 80 80 C0 D0 D0 D4
4950: A0 A0 A0 A8 AA AA AA AA
4958: 95 95 95 95 95 85 85 81
4960: AA AA AA AA AA AA AA AA
4968: 81 81 81 81 81 81 81 81
4978: D4 D0 D0 D0 D0 D0 D0 D0
4980: 82 82 82 A2 A2 AA AA AA
4988: D5 95 95 85 85 81 81 80
```

Para gravar o banco na fita use:

```
4000.5000 W
```

No TK-2000 use:

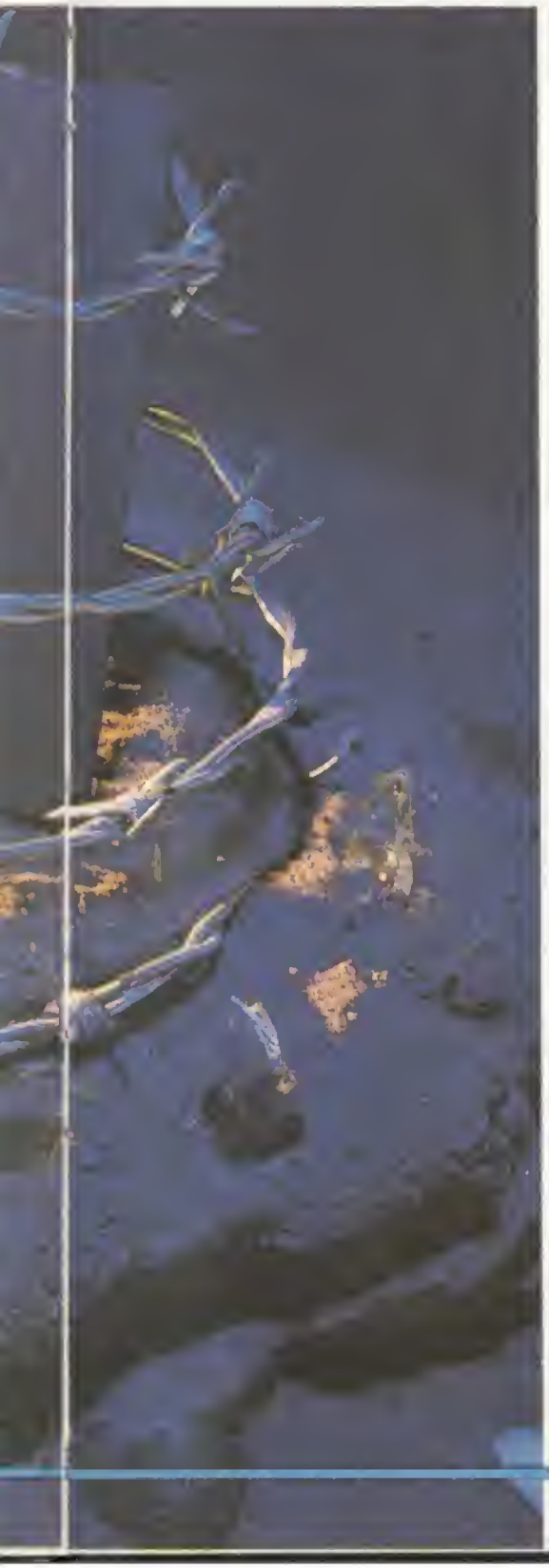
```
A000.B000 W "nome"
```

Isto deve ser digitado ainda dentro do monitor. Quem tiver unidade de disquete pode utilizar o gerador para gravar o banco.

PROTEJA SEUS PROGRAMAS

Você quer proteger seus programas contra a "pirataria" ou a mera curiosidade? Veja aqui as técnicas utilizadas pelos profissionais na montagem de travas e armadilhas.

- O QUE SE PODE FAZER PARA PROTEGER PROGRAMAS BASIC
- PROGRAMAS AUTOCARREGÁVEIS: O QUE SÃO E O QUE FAZEM
- PROGRAMAS INTERDEPENDENTES
- COMO DESATIVAR OS COMANDOS **SAVE** E **LIST**
- PEQUENAS ARMADILHAS PARA O SEU COMPUTADOR



Ainda que você tenha muita imaginação e domine as técnicas necessárias para montar as mais fantásticas armadilhas, seu programa nunca estará totalmente protegido dos olhares curiosos. Mas, combinando determinados recursos — alguns mais simples, outros nem tanto —, você pode dificultar tanto o trabalho do "pirata" que ele se sentirá tentado a abandonar sua tarefa.

Nunca é demais dar a um programa um toque profissional, sobretudo depois de um grande esforço para deixá-lo "redondinho". Nesse sentido, você pode fazer duas coisas: melhorar a apresentação do programa e providenciar-lhe uma proteção razoável, de modo que suas técnicas especiais fiquem, ao menos, obscurecidas para o público.

Algum cuidado no acabamento é essencial se você pensa em comercializar seu programa — especialmente se pretende negociá-lo com uma *software-house*. Os recursos que você deve utilizar para dar-lhe uma melhor apresentação foram discutidos anteriormente; assim, focalizaremos aqui as técnicas disponíveis para garantir alguma proteção ao seu trabalho.

A proteção de um programa BASIC depende de armadilhas colocadas dentro do próprio programa. Os escritos em código de máquina podem receber um tratamento bem mais sofisticado.

Não existem restrições de nenhum tipo quanto ao emprego de armadilhas para evitar as cópias "piratas" ou a listagem de um programa por curiosos. Quantas delas serão colocadas no programa é questão para você decidir.

De qualquer maneira, convém sempre se lembrar de que não se conhecem métodos absolutamente seguros de evitar que um programa seja copiado. Muitas pessoas encaram os programas protegidos como um desafio e não desistem até que ele seja vencido. Outras se empenharão em listar o programa para examiná-lo, adaptá-lo às suas necessidades ou, simplesmente, aprender.

PRIMEIROS PASSOS

Um dos métodos para proteger programas consiste em introduzir neles várias armadilhas simples. Elas não impedem que alguém com conhecimento da máquina abra o programa, mas tornam a tarefa bem mais trabalhosa. Infelizmente, esse método dificulta também a elaboração do programa. Além disso, ele complica a depuração de erros, já que as armadilhas podem estar ativas durante a execução do programa.

As travas mais simples são as que introduzem no programa mudanças que tornam impossível o uso dos comandos **SAVE**, **LIST** e outros. Como elas só se ativam após a execução do programa, não criam os problemas de depuração mencionados acima.

AUTOCARREGAMENTO

Pode-se obter uma proteção bem melhor fazendo com que o programa seja executado automaticamente depois de carregado. No processo normal de carregar um programa (**LOAD**), os comandos são digitados no modo direto. Mas eles podem ser chamados por um programa à parte, denominado programa de autocarregamento (*bootstrap*). Este e escrito em BASIC ou código de máquina, dependendo das tarefas específicas que deve executar. Na sua forma mais simples, pode ser algo como:

```
10 LOAD "NOME PROXIMO PROGRAMA"
```

A execução desse pequeno programa carregaria para a memória do micro o programa cujo nome fosse especificado. Obviamente, o uso de uma linha como esta é bem específico. Mas esse tipo de programa pode ser empregado para fazer muito mais, sendo bastante útil para programação suplementar — como montar páginas-títulos, gráficos etc.

Estas incluem poderosas armas de proteção de um programa BASIC, que alteram determinados comandos do sistema. No nosso caso, o programa de auto-carregamento funciona como um inicializador do sistema que permite que seu programa seja carregado e executado.

Vários tipos de programas comerciais empregam *bootstraps*, muitas vezes gravados na forma de programas patrocinados, onde cada parte é carregada em seqüência. Lembre-se de que, normalmente, um programa BASIC apaga o anterior ao ser carregado. Assim, se você utilizar essa técnica, carregue antes os módulos em linguagem de máquina e, depois, o BASIC.

Com programas patrocinados, a proteção pode ser determinada por um certo grau de interdependência entre um arquivo (parte do programa) e outro. Neste caso, um arquivo checa um valor de memória determinado por outro. Pode-se ainda acrescentar um programa que cheque um arquivo de dados especial colocado após o programa principal. Em ambas as alternativas, a falta de qualquer um dos módulos provoca um erro e impede que o programa funcione.

Os bootstraps oferecem mais uma vantagem: com sua utilização, o tempo de carregamento dos programas na memória diminui, porque o código de máquina e os dados podem ser colocados diretamente na memória. Esse programa dispensa, assim, o uso de declarações **DATA** do BASIC, que só funciona após o programa começar a ser executado.

AUTO-EXECUÇÃO

A utilização de um programa de autocarregamento não é muito simples no TRS-Color (usando-se o cassete). Esse computador não dispensaria chamadas especiais do sistema, o que não é possível com o BASIC. No Spectrum, porém, para executar um programa a partir de outro basta usar o comando apropriado em algum lugar do programa de autocarregamento.

```
990 LOAD "NOME DO PROXIMO PROGRAMA"
```

Grave (**SAVE**) este segundo programa usando **SAVE "NOME DO PROXIMO PROGRAMA" LINE 1** — ou o número de linha que represente o início do programa. Escolhendo um número de linha maior, você poderá incluir avisos ou dados para serem lidos (**PEEK**) e checados em linhas REM anteriores à linha inicial de execução.

No MSX também é muito simples executar um programa a partir de outro: basta usar o comando **LOAD** com a opção R. Mas, para que tudo dê certo, o programa a ser executado deve estar gravado no formato ASCII, isto é, com a instrução **SAVE**. Na sua forma mais simples, o programa seria:

```
100 LOAD "NOME DO PROGRAMA",R
```

As técnicas que seguem valem apenas para micros com disquete.

Veremos, primeiro, como montar um programa de execução automática. O método mais simples consiste em introduzir no programa, que é automaticamente carregado e executado ao se ligar o computador, uma linha do tipo **PRINT CHR$(4); "RUN PROGRAMA"**. Mas a utilização de um arquivo-texto é mais elegante, e exige apenas que o programa com que o disquete foi inicializado contenha esta linha:

```
10 PRINT CHR$(4);"EXEC AUTORUN"
```

AUTORUN é um arquivo-texto que será executado como se as instruções estivessem sendo digitadas a partir do teclado. Para criá-lo, digite e execute este programa:

```
10 D$ =  CHR$ (4)
20  PRINT D$;"OPEN AUTORUN"
30  PRINT D$;"WRITE AUTORUN"
40  PRINT "NOMON C,I,O"
50  PRINT "RUN BP35"
60  PRINT D$;"CLOSE"
```

Com esse arquivo, qualquer tentativa de interromper o programa será interpretada como um erro. E isso será aproveitado por uma rotina de tratamento de erros no programa principal.

Digite este programa e salve-o, por exemplo, com o nome de BP35:

```
10  HOME
20  PRINT "ALO    ";
30  GOTO 20
```

Em seguida, digite "**EXEC AUTORUN**". O programa será executado pelos comandos do arquivo **AUTORUN**. Se inicializou o disquete como sugerimos, seu programa BP35 terá execução automática toda vez que o computador for ligado com esse disquete no drive 1.

O USO DE VARIÁVEIS DO SISTEMA

A auto-execução não basta para afastar os curiosos. É necessário utilizar outros recursos para evitar que os programas sejam interrompidos, listados ou alterados. Um desses recursos consiste em alterar o modo como o sistema reage quando se faz uma chamada específica — por exemplo, a sub-rotina de listagem (LIST).

Cada computador tem um sistema operacional e um interpretador BASIC. A maior parte da informação é guardada na memória apenas para leitura — ROM —, mas parte dela é transferida para a memória de acesso aleatório — RAM —, quando se liga o computador. Esta informação pode ser modificada para alterar o funcionamento do sistema, permitindo que se incorporem aos programas métodos sofisticados de proteção.

Como estaremos interferindo na própria forma de trabalho do computador, precisaremos ter em mãos uma listagem completa das variáveis e rotinas do sistema, e um bom mapa de memória. Veja o seu manual de referência.

A transferência de parte da informação do sistema da ROM para a RAM, possibilita a alteração do valor de algumas variáveis. Estando na RAM, esta informação torna-se vulnerável a qualquer modificação que se queira fazer.

Um tipo especial de variável do sistema é o apontador da RAM. Este consiste, em geral, de dois endereços consecutivos que guardam o endereço de uma sub-rotina específica. Se alterarmos tal endereço, o sistema será redirecionado sempre que esta sub-rotina for chamada. Os melhores apontadores para a proteção de programas são os que se referem aos comandos **LIST**, **SAVE**, **INTERRUPT** e **RESET**. Os últimos comandos fogem ao escopo deste artigo.

ESTUDO DE CASOS

É muito difícil dar total proteção aos programas, especialmente se se quer evitar um maior envolvimento com código de máquina. Os métodos para cada máquina diferem muito, já que os sistemas operacionais também são bastante diferentes. Mas vale a pena analisar algumas possibilidades.

Uma das maneiras mais simples de marcar um programa consiste em colocar uma mensagem que não possa ser retirada. Para reforçar a proteção, convém acrescentar uma rotina que verifique a presença da marca.

Inicialmente, encontre o endereço da área reservada para programas em BASIC. Esse endereço (variável de sistema **PROG**) é armazenado nas posições 23635 e 23636 e pode ser determinado

pela instrução: **PRINT PEEK 23635 + 256 * PEEK 23636**. Conhecendo o valor de **PROG**, você pode colocar qualquer número em **PROG + 1**, alterando o número da primeira linha do programa.

O segredo é deixar a linha com o número 0, já que não é possível apagar a linha 0. Suponhamos que a primeira linha do programa seja:

```
10 REM (C) NOVA CULTURAL 1986
```

Digite este comando:

```
POKE(PEEK 23635 + 256 * PEEK
25636) + 1,0
```

Pronto! A linha 10 virou linha 0.

O mesmo pode ser feito pelo programa de autocarregamento. Mas, tratando-se de um programa auto-executável, a maneira óbvia de interrompê-lo é teclando **<BREAK>**, que coloca uma mensagem na parte de baixo da tela. Suponhamos, porém, que esta não aceite a mensagem. Na lista das variáveis de sistema, vê-se que **DFSZ** (no endereço 23659) armazena o número de linhas da parte inferior da tela (geralmente 2). Alterando este valor para 0, o computador detectará um erro assim que a mensagem tentar aparecer na tela.

Como não se pode entrar tais instruções no modo direto, digite:

```
10 POKE 23659,0
20 PRINT AT 5,5; RND
30 GOTO 20
```

Execute o programa. Se pressionar **<BREAK>**, provocará um erro.

Quando usar este método, lembre-se de que, se seu programa contiver situações que produzem mensagens como **INPUT?** ou **scroll?**, o erro será inevitável. Assim, para entrada de dados, use **INKEY$**.

O "pirata" pode evitar que um programa se auto-execute, carregando-o com **MERGE**. Mas será impossível fazê-lo se o programa for gravado em código. As variáveis do sistema, o programa e a memória livre devem ser gravados acima do buffer da impressora.

A gravação começa com **CODE 23552**, que é o início da área das variáveis do sistema. O número de bytes é N-23552, onde N é qualquer número maior que **STKEND** (o endereço do início do espaço livre). Obtém-se este endereço com **PEEK 23653 + 256*PEEK 23654**.

Coloque estas linhas no início do seu programa:

```
1 SAVE "NOMEPROG" CODE 23552,
  N-23552
2 POKE 23659,0
```

Agora, iguale N ao endereço descrito acima e digite **GOTO 1**. O programa será gravado. O comando **LOAD "NOMEPROG" CODE** fará com que o programa seja executado.

Vimos como proceder para que um programa inicial carregue e execute o programa principal. Mas isso não é suficiente. Um simples **<CTRL><STOP>** permite que se interrompa o programa para que ele seja listado.

No MSX, pode-se evitar uma interrupção desse tipo com facilidade. A idéia é direcionar o programa para uma sub-rotina especial toda vez que se tentar uma interrupção com **<CTRL><STOP>**. Para isso, utilizam-se as instruções **STOP ON**, que fazem com que o BASIC verifique a todo instante se uma interrupção foi tentada, e **ON STOP GOSUB XXXXX**, que desvia o programa para uma determinada linha em caso positivo. Digite o seguinte:

```
10 ON STOP GOSUB 1000
20 STOP ON
30 CLS
40 LOCATE 10,12:PRINTRND(1):GOT
O 40
1000 P=P+1
1010 CLS:LOCATE 10,10:PRINT"Não
 seja curioso!":BEEP:FOR S=1 TO
 1000:NEXT
1020 IF P=3 THEN NEW:END
1030 CLS:RETURN
```

Não se esqueça de gravar o programa antes de executá-lo. Depois, pare-o se puder!

Você já sabe como proceder para que seu programa seja automaticamente executado; veja agora como impedir que o interrompam, ou seja, como impossibilitar sua listagem. Para isso, fazemos com que toda tentativa de interrupção — por meio do **<CTRL><C>** ou do **<CTRL><RESET>** — seja interpretada como um erro, desviando a execução para uma rotina especial. Para que o micro interprete o **<CTRL><RESET>** como um erro (o **<CTRL><C>** é normalmente interpretado como tal), adicione as linhas seguintes ao programa do arquivo **AUTORUN**.

```
45  PRINT "POKE 40286,35"
46  PRINT "POKE 40287,216"
```

Elimine o **AUTORUN** já existente e execute o programa. O novo arquivo contém os dois **POKE** que mudam o comportamento do computador em relação ao **<RESET>**. Mas, para que possamos tirar proveito dessa mudança, uma rotina de erros deve ser inserida no programa principal:

```
5  ONERR  GOTO 1000
1000 P = P + 1
1010  IF P = 3 THEN  PR# 6
1020  PRINT  CHR$ (7);: RESUME
 NEXT
```

Grave o programa alterado. Desligue o computador e ligue-o novamente. Tente parar o programa...

Pode ocorrer, porém, que o curioso se lembre de olhar para o diretório do disco e carregar o seu programa diretamente, sem executá-lo; assim, poderá listá-lo. Para confundi-lo, inclua no nome do programa, no momento de gravar (**SAVE**), alguns caracteres de controle, que ficarão invisíveis. Para fazer isto, basta digitar junto do nome do programa um ou mais **<CTRL><letra>**, onde a letra pode ser A, B etc. Evite o C e o X.

ZX-81: EDIÇÃO DE PROGRAMAS

- COMO EDITAR LINHAS DE UM PROGRAMA BASIC
- OS MOVIMENTOS DO CURSOR
- O COMANDO EDIT
- CONSOLIDE AS MODIFICAÇÕES

Os micros da linha ZX-81 possuem bons recursos de edição. Com o comando EDIT e as teclas de movimentação do cursor, você pode alterar ou duplicar rapidamente uma linha de programa.

Devido à sua natureza "conversacional", todos os interpretadores da linguagem BASIC incluem alguns recursos de edição de programas, ou seja, técnicas de entrada e alteração das linhas que o compõem.

Como vimos em artigos anteriores sobre edição de programas em outras linhas de microcomputadores, o primeiro interpretador BASIC, desenvolvido na década de 60 na Universidade de Dartmouth, nos EUA, fixou os recursos mais elementares de edição: numeração, inserção, apagamento e listagem de linhas. Estes recursos, adotados posteriormente em todos os dialetos BASIC que surgiram, não davam ao programador a possibilidade de alterar caracteres. Assim, para corrigir um único caractere errado, ele teria que digitar toda a linha novamente.

Para poupá-lo desse trabalho, desenvolveram-se comandos como o **EDIT**, presentes nos micros da linha Sinclair, TRS-80 e TRS-Color. O comando **EDIT** dos micros da linha ZX-81 opera em uma linha de cada vez.

DIGITAÇÃO DE PROGRAMAS

No ZX-81, cada linha é digitada de uma vez, precedida de um número.

A parte inferior da tela do ZX-81 destina-se à digitação de linhas. Ao ser ligado, o computador exibe um cursor de texto — um retângulo em vídeo inverso — contendo a letra K. Esta é a chamada *linha de edição* da tela. À medida que você vai digitando a linha, por meio do teclado, este cursor vai se deslocando à frente dos caracteres. Durante o deslocamento, ele pode mudar de tipo (passar para tipo L, tipo G, tipo F etc.).

Quando se pressiona a tecla <**ENTER**> ou <**RETURN**>, a linha é completada e inserida no ponto certo do programa. Enquanto isso não ocorre, pode-se modificar a linha à vontade.

Três teclas de controle são utilizadas para se proceder às modificações: as teclas com as flechas para a esquerda e para a direita (← e →), que devem ser acionadas simultaneamente com a tecla <**SHIFT**>, e a tecla de apagamento (chamada de **DELETE** ou **RUBOUT**, conforme a marca do computador, e que, no ZX-81, corresponde à pressão simultânea das teclas 9 e <**SHIFT**>).

Ao ser acionada, a tecla de apagamento pode eliminar um caractere ou uma palavra-chave do BASIC de uma vez. O restante da linha se desloca para a esquerda, de modo a preencher os claros deixados.

O ZX-81 dispõe do modo de inserção automático, ou seja, se digitarmos caracteres quando o cursor estiver no meio de uma linha, estes serão inseridos, por intermédio do deslocamento do restante da linha para a direita. Para sobrepor caracteres (escrever sobre caracteres já existentes), é necessário primeiro apagá-los.

O COMANDO EDIT

Uma vez pressionada a tecla <**ENTER**>, a linha digitada passa a fazer parte do programa. Se, depois disso, você digitar apenas o seu número, e pressionar <**ENTER**> de novo, ela será apagada. Por outro lado, se você digitar seu número, seguido de novos comandos ou instruções, estes substituirão a linha existente no programa.

Um recurso de edição que faz falta no ZX-81 é o comando **DELETE** (ou, ainda, **DEL**, em alguns computadores), que apaga grupos de linhas de uma vez só. Os usuários desse micro precisam sempre digitar os números de cada uma das linhas a serem apagadas, um a um, seguidos de <**ENTER**>.

Para modificar uma linha já existente, sem precisar redigitá-la inteiramente, pode-se recorrer ao comando **EDIT**. Para isso é necessário entender o conceito de *cursor de linhas*.

Se você olhar para uma listagem de programa na tela, perceberá que existe sempre um retângulo em vídeo inverso, contendo o sinal de maior (>) em seu interior, e que aponta para uma das linhas na tela. Esse retângulo é o cursor de linhas. Para deslocá-lo, basta pressionar uma ou mais vezes uma das teclas com a flecha para cima ou para baixo (↑ ou ↓).

Agora, se você pressionar simultaneamente as teclas <**SHIFT**> e **1**, a linha apontada pelo cursor de linhas aparecerá na linha de edição.

Você poderá usar, então, todos os procedimentos de edição: movimentação do cursor de texto, apagamento, inserção etc. Tudo funciona da mesma maneira, até que você pressione a tecla <**ENTER**>. Ao fazê-lo, a linha modificada irá substituir a que foi chamada anteriormente pelo **EDIT**.

DUPLICAÇÃO DE LINHAS

A duplicação de linhas é um dos recursos mais interessantes do ZX-81. O número da linha pode ser editado normalmente. Assim, se apenas ele for modificado, a linha com o número original mantém-se no programa, e a mesma linha, com nova numeração, é inserida no ponto correto.

Por outro lado, se, além do número da linha, também o seu conteúdo for modificado, uma linha diferente será automaticamente gerada.

Portanto, para que uma linha possa ser editada pelo comando **EDIT**, ela precisa estar listada na tela (use o comando **LIST número**, para isto). Desloque o cursor de linha até ela, e pressione o comando **EDIT**.

Para agilizar o processo de edição de várias linhas de um programa extenso, você pode utilizar um pequeno truque. Em vez de pressionar repetidas vezes as teclas de controle do cursor de linhas, até chegar à linha que você deseja, digite o comando **LIST**, seguido do número dessa linha. Isto a colocará no alto da tela, com o cursor de linhas automaticamente apontado para ela. Basta, então, pressionar o comando **EDIT**.

SÍMBOLOS GRÁFICOS NO MSX

Os micros da linha MSX dispõem de caracteres gráficos que podem ser entrados diretamente pelo teclado ou usados dentro de um programa. Veja como explorar este recurso especial.

Os microcomputadores da linha MSX são extremamente versáteis do ponto de vista da programação gráfica. Em artigos anteriores, vimos como as teclas gráficas (**SCREEN**) podem ser programadas, em média e alta resolução, por meio de instruções poderosas como **LINE**, **CIRCLE**, **PAINT**, **PSET** etc.

O MSX apresenta, ainda, um recurso gráfico adicional que é pouco explorado. São os *caracteres gráficos*, disponíveis para o programador por meio de dois procedimentos: entrada direta pelo teclado e inserção através das funções do BASIC **CHR$** e **STRING$**.

O que são caracteres gráficos? Como você já sabe, os caracteres que aparecem no vídeo têm códigos numéricos inteiros, entre 0 e 255; cada caractere corresponde, portanto, a um byte da memória do vídeo. Parte dessa codificação está convencionada internacionalmente pelo chamado código ASCII, que vai de 32 a 126. Os códigos de 0 a 31 são normalmente utilizados para funções de controle do vídeo, e dependem do tipo de computador. O mesmo acontece com os códigos de 127 a 255. Nesta faixa, cada fabricante usou os códigos de uma maneira, em geral para acomodar caracteres em outros idiomas que não o inglês (é o caso do MSX, existente no Brasil) ou para especificar caracteres gráficos. Estes tanto podem ser tipos especiais, para a imagem de um rosto sorrindo ou uma nota musical, como blocos gráficos formando linhas, ângulos e cantos.

GRÁFICOS DA ROM

Tais caracteres são também chamados de *gráficos da ROM*, pois já vêm pré-programados. No MSX, os caracteres especiais ocupam duas faixas da tabela de caracteres:

- a faixa de 1 a 31
- a faixa de 144 a 254

Nestas faixas, os caracteres gráficos propriamente ditos encontram-se dispersos em vários pontos da tabela, entremeados com caracteres de outros idiomas, com o alfabeto grego, símbolos matemáticos etc.

Os caracteres gráficos que mais nos interessarão neste artigo são os blocos empregados na formatação de tabelas, de formulários de entrada ou na composição de quaisquer outros desenhos formados por linhas retas. A utilização de caracteres gráficos, nesses casos, tem a vantagem de não complicar a programação, misturando tela gráfica com texto, o que nos permite recorrer a comandos mais simples, como o **LOCATE**, o **PRINT**, o **INPUT** etc.

ENTRADA PELO TECLADO

Assim como os caracteres convencionais do ASCII (demarcados sobre as teclas do microcomputador), os caracteres especiais e gráficos podem ser entrados pelo teclado. Para isso, pressiona-se simultaneamente uma ou mais de três teclas de função — <**GRAPH**>, <**CODE**> ou <**SHIFT**>—, com outra tecla normal do teclado. Com isso, o caractere desejado aparece diretamente na tela (por exemplo, dentro de uma cadeia alfanumérica).

Com o programa abaixo, você verá como entrar caracteres pelo teclado. Ele desenha uma tabela simples na tela, usando blocos gráficos, e depois coloca os nomes digitados pelo usuário nas linhas da tabela.

```
200 CLS
210 PRINT"                    "
220 PRINT" No.  Nome          "
230 PRINT"                    "
240 FOR I=1 TO 5
250 PRINT"                    "
260 PRINT"                    "
270 NEXT I
280 PRINT"                    "
290 PRINT"                    "
300 FOR I=1 TO 6
310 LOCATE 0,21:PRINT STRING$(3
0,32)
320 LOCATE 0,21:LINE INPUT "NOM
E: ";N$
330 LOCATE 1,(I*2)+1:PRINT USIN
G "##";I;
340 LOCATE 7,(I*2)+1:PRINT LEFT
$(N$,10);
350 NEXT I
360 LOCATE 0,21:PRINT "FIM
 ":END
```

Os traços são teclados na seguinte seqüência:

Linha 210: <**GRAPH**> R
<**GRAPH**> - (4 vezes)
<**GRAPH**> T
<**GRAPH**> - (11 vezes)
<**GRAPH**> Y
Linha 220: <**SHIFT**> <**GRAPH**>
Linha 230: <**GRAPH**> F
<**GRAPH**> - (4 vezes)
<**GRAPH**> G
<**GRAPH**> - (11 vezes)
<**GRAPH**> H

Linha 250: como a linha 220
Linha 260: como a linha 230
Linha 280: como a linha 220

Linha 290: <**GRAPH**> V
<**GRAPH**> - (4 vezes)
<**GRAPH**> B
<**GRAPH**> - (11 vezes)
<**GRAPH**> N

Há duas desvantagens nesta maneira de entrar caracteres gráficos. Primeiro, as teclas não têm qualquer marcação que auxilie o usuário a achar o gráfico correto; assim, é preciso consultar o manual, o que torna o processo bastante trabalhoso. Em segundo lugar, o programa não pode ser listado em uma impressora não gráfica, ou que não seja própria para o MSX.

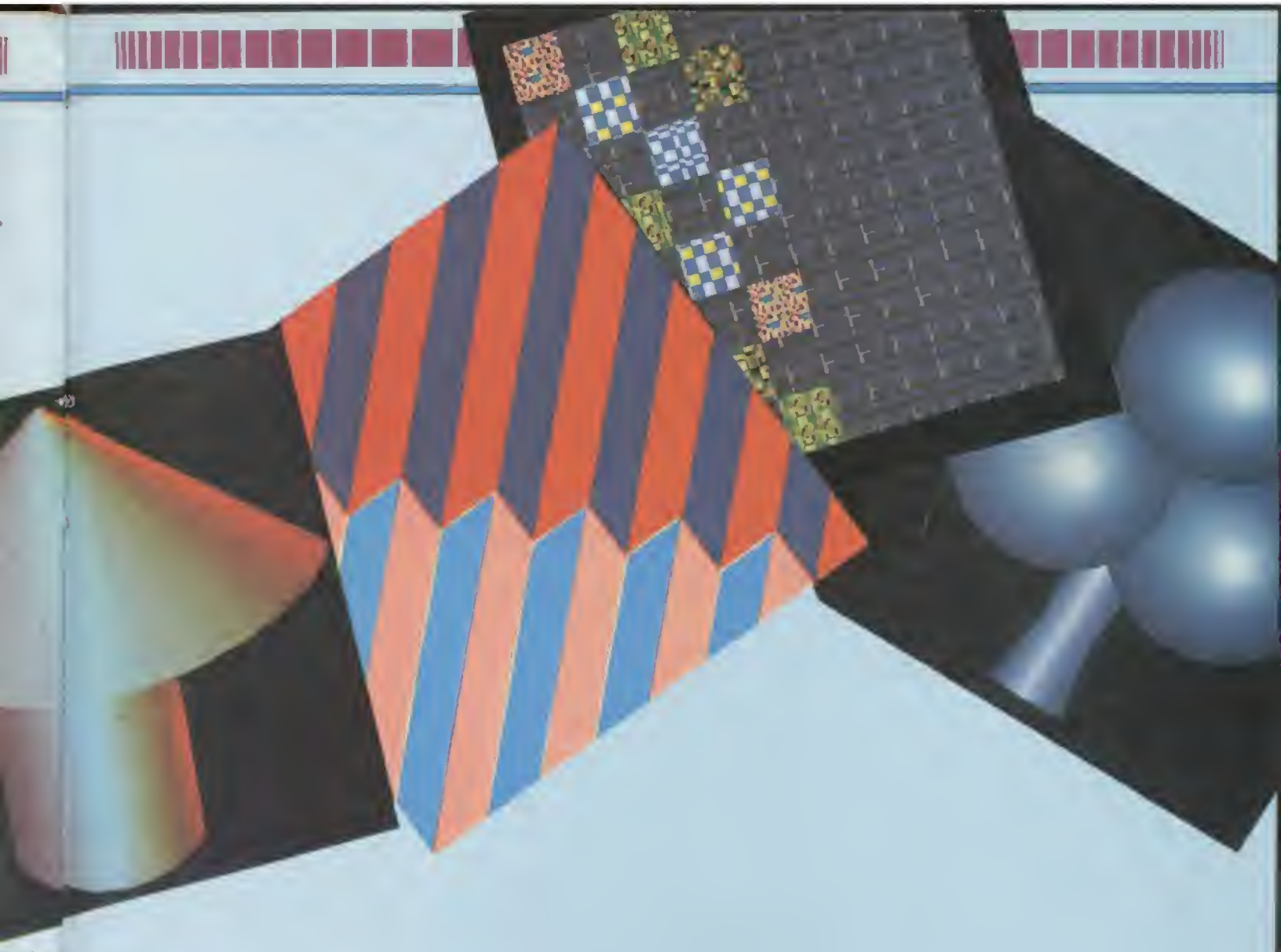
CARACTERES GRÁFICOS NO PROGRAMA

Para especificar caracteres gráficos dentro de um programa, sem precisar digitá-los diretamente, podemos usar as funções **CHR$** e **STRING$**: com o número de código do caractere desejado, elas permitem que os caracteres normais, especiais e gráficos com códigos na faixa de 32 a 254 sejam impressos na tela a partir de um programa. Por exem-

CARACTERES GRÁFICOS DO MSX

Código	Teclas	Caractere	Código	Teclas	Caractere
1	GRAPH '		192	GRAPH U	
2	SHIFT GRAPH '		193	SHIFT GRAPH D	
3	SHIFT GRAPH "		194	GRAPH I	
4	SHIFT GRAPH C		195	SHIFT GRAPH O	
5	GRAPH "		196	GRAPH A	
6	GRAPH C		197	SHIFT GRAPH I	
8	SHIFT GRAPH 9		198	GRAPH J	
9	GRAPH 0		199	GRAPH D	
10	SHIFT GRAPH 0		200	GRAPH L	
11	GRAPH M		201	SHIFT GRAPH L	
12	SHIFT GRAPH M		202	SHIFT GRAPH J	
13	GRAPH '		203	SHIFT GRAPH Q	
14	SHIFT GRAPH '		204	GRAPH Q	
15	GRAPH Z		205	GRAPH E	
16	SHIFT GRAPH G 13		206	SHIFT GRAPH E	
17	GRAPH B		207	GRAPH W	
18	GRAPH T		208	SHIFT GRAPH W	
19	GRAPH H		209	SHIFT GRAPH S	
20	GRAPH F		210	GRAPH S	
21	GRAPH G		211	SHIFT GRAPH N	
22	SHIFT GRAPH \		212	SHIFT GRAPH F	
23	GRAPH -		213	SHIFT GRAPH V	
24	GRAPH R		214	SHIFT GRAPH H	
25	GRAPH Y		215	SHIFT GRAPH P	
26	GRAPH V		219	GRAPH P	
27	GRAPH N		220	GRAPH O	
28	GRAPH X		221	GRAPH K	
29	GRAPH /		222	SHIFT GRAPH K	
30	GRAPH \		223	SHIFT GRAPH U	
31	SHIFT GRAPH -		248	SHIFT GRAPH Z	
169	SHIFT GRAPH R		249	SHIFT GRAPH C	
170	SHIFT GRAPH Y		250	SHIFT GRAPH X	
188	SHIFT GRAPH C		254	SHIFT GRAPH A	

O repertório de caracteres gráficos do MSX é muito variado: inclui desde tipos especiais, como uma nota musical ou um rosto sorrindo, até blocos gráficos compondo linhas, ângulos e cantos. Para entrá-los pelo teclado, oriente-se pelo quadro de referência ao lado. Como você pode observar, será sempre necessário pressionar, simultaneamente, uma ou mais teclas de função e uma tecla normal.



plo, para imprimir na tela a letra grega beta, digite:

```
PRINT CHR$(225)
```

Ou, então, para traçar uma linha reta de dupla espessura na tela, use:

```
PRINT STRING$(32,197)
```

A função **STRING$ (n,c)** retorna uma cadeia de **n** caracteres com código **c**. Já os caracteres gráficos que se encontram na faixa de 1 a 31 não podem ser impressos da mesma maneira, pois estes códigos têm funções de controle do vídeo. Para usá-los com a função **CHR$**, é necessário "avisar" o computador que o código será usado como gráfico. Para isso, são empregados dois bytes: **CHR$(1)**, seguido de **CHR$(n+64)**, onde **n** é o código do caractere gráfico na tabela. Por exemplo, **PRINT CHR$(1);CHR$(75)** mostrará na tela o símbolo masculino.

Esse método funciona também para a faixa de códigos gráficos que vai até 191. O programa abaixo mostra na tela uma tabela de correspondência:

```
10 KEY OFF:CLS:I=0
20 FOR N=1 TO 6
30 I=I+1:IF I>191 THEN GOSUB 10
0:END
40 PRINT USING "### ";I;
50 PRINT CHR$(1)+CHR$(I+64);" "
;
60 NEXT N
70 PRINT:PRINT
80 L=L+1:IF L<10 THEN 20
90 GOSUB 100:L=0:GOTO 20
100 LOCATE 0,22
110 PRINT "Pressione qualquer t
ecla p/continuar"
120 IF INKEY$="" THEN 120
130 CLS:RETURN
```

Se você substituir três linhas — 10, 30 e 50 —, o programa servirá também para mostrar os códigos gráficos de 128 a 254.

```
10 KEY OFF:CLS:I=127
30 I=I+1:IF I>254 THEN GOSUB
   100:END
50 PRINT CHR$(I);" ";
```

Caso pretenda utilizar um caractere gráfico da faixa de 1 a 31 com freqüência num programa, armazene-o em uma variável alfanumérica, como mostra o exemplo abaixo. Os quatro símbolos dos naipes do baralho são armazenados em **N$** (e seus nomes em **E$**):

```
10 CLS
20 FOR I=1 TO 4
30 READ E$(I)
40 N$(I)=CHR$(1)+CHR$(I+66)
50 PRINT N$(I),E$(I)
60 NEXT I
70 DATA Copas,Ouros,Paus,Espada
s
```

EFEITOS SONOROS NO SPECTRUM

Os recursos sonoros do Spectrum são limitados, se considerarmos apenas o comando SOUND. Porém, com código de máquina, poderemos produzir vários sons, de sirene a tiros de laser.

Em geral, o endereçamento da memória é uma função do microprocessador. Na maioria dos computadores domésticos, quando queremos usar um aparelho externo — como uma impressora, um televisor ou mesmo o teclado do computador — temos que fazê-lo por meio de uma posição de memória que está ligada a uma porta de saída. Porém, com micros que usam o Z-80, como o Spectrum, por exemplo, pode-se ter acesso direto às portas, tanto em BASIC, com os comandos **IN** e **OUT**, como em código de máquina, com **in** e **out**.

AFINAL, O QUE É UMA PORTA?

Uma porta é um canal de comunicação entre o microprocessador e o mundo externo. Este inclui o teclado e tudo o que é periférico ao microprocessador, à RAM e à ROM.

Já tivemos oportunidade de ver como **in** é usado para dar acesso ao teclado. Os comandos **OUT** e **out** funcionam de modo semelhante, só que, em vez de receber dados, enviam-nos aos periféricos. Ambos têm acesso bem mais versátil ao alto-falante do que o comando BASIC **SOUND** e podem ser usados, também, para controlar a moldura da tela. Entretanto, como o **OUT** é excessivamente lento, consideraremos apenas o comando **out**.

Para gerar sons com **out** precisamos da velocidade da linguagem de máquina. Eles são obtidos por meio do movimento do cone do alto-falante para fora e para dentro. Movimentando-o uma vez, produzimos um clique, do tipo que se ouve ao ligar um aparelho de som. O truque está em repetir o movimento muitas vezes, e rapidamente.

Teremos, assim, uma sucessão de cliques, que provocam um efeito similar a um zumbido. Quanto mais veloz for o movimento de vaivém do cone, mais agudo será o som resultante.

A rotina Assembler apresentada a seguir produz um som cuja tonalidade vai aumentando. Digite **CLEAR 64599** e depois as linhas:

```
10 REM org 64600
20 REM ld a,(23624)
30 REM rrca
```

- A PORTA 254
- INSTRUÇÕES SHIFT E ROTATE
- O USO DO ALTO-FALANTE
- LAÇOS DE PAUSA
- ACERTE O TOM
- MODIFICAÇÃO DA MOLDURA
- A ESCOLHA DA COR
- ACERTO DO TEMPO
- SINTONIA FINA
- COMO DOBRAR A FREQÜÊNCIA

```
 40 REM rrca
 50 REM rrca
 60 REM ld b,0
 70 REM loop push bc
 80 REM xor 16
 90 REM out 254,a
100 REM pause nop
110 REM nop
120 REM djnz pause
130 REM pop bc
140 REM djnz loop
150 REM ret
```

A porta 254, que controla o alto-falante, controla também a moldura da tela. Para que um comando **out** não modifique a moldura ao produzir um som, utiliza-se uma rotina em código destinada a evitar o problema.

Inicialmente, coloca-se no acumulador o valor da variável do sistema situado na posição 23624 da memória. As cores do Spectrum têm códigos que vão de 0 a 7. Normalmente, os bits que controlam a cor são os bits três, quatro e cinco. Contudo, quando estamos controlando a moldura via porta 254, os bits zero, um e dois encarregam-se da modificação das cores.

Assim, para se ter a certeza de que a cor da moldura não será modificada pelo efeito sonoro, os bits três, quatro e cinco devem ser deslocados três posições para a direita (para o lugar dos bits zero, um e dois).

SHIFT E ROTATE

O deslocamento dos bits pode ser executado por vários comandos. Utilizamos aqui **rrca** (*rotate right with carry, on the acumulator*, expressão em inglês que significa rotação para a direita com "vai um" no acumulador). Essa instrução movimenta todos os bits que estão no acumulador uma posição para a direita. É chamada de rotação porque coloca o conteúdo do bit zero no bit sete, fazendo-o "dar a volta", por assim dizer. Um comando SHIFT — deslocamento — moveria todo o conteúdo do acumulador um bit para a direita, mas preencheria o bit de reserva com zero. Seu emprego é mais adequado a operações aritméticas: um SHIFT para a esquerda multiplica um número por dois, enquanto um SHIFT para a direita o divide por dois.

Aqui, porém, utilizamos uma rotação porque queremos deslocar apenas três dos bits. O conteúdo dos demais não nos interessa. O comando **rrca** ainda copia o conteúdo do bit zero no bit carry — "vai um". Mais uma vez, não importa o valor do carry.

Para deslocar os bits de cor da moldura três posições para a direita, emprega-se **rrca** três vezes, o que equivale a dividir a cor da moldura por oito. Mas quando a cor — normalmente um número entre 0 e 7 — se encontrava nos bits três, quatro e cinco, estava multiplicada por oito.

Agora, quando usarmos o comando **out**, ele especificará a mesma cor de moldura que havia antes, e nenhuma mudança ocorrerá.

ACERTE OS CONTADORES

O registro B funciona como um contador duplo. Para começar, colocaremos 0 nele, e este valor será guardado na pilha por **push bc**. O registro B não pode ser guardado sozinho na pilha, já que os comandos de **push** e **pop** agem apenas em pares de registros. Precisamos, assim, guardar B juntamente com C. Como não usaremos o registro C, isso não afetará o programa.

COMO PRODUZIR SONS

O bit quatro da porta 254 controla o alto-falante. Alternando o valor desse bit, movimenta-se o cone do alto-falante, produzindo som.

Alterna-se o bit quatro por meio da operação lógica "ou exclusivo". Fazemos um "ou exclusivo" entre o valor do acumulador e dezesseis. Assim, se o bit quatro estiver ligado, valendo 1, ele será desligado, passando a valer 0 — ou vice-versa.

O **out 254,a** envia o conteúdo do acumulador via porta 254. Em muitos Assemblers comerciais os comandos **out** e **in** precisam de colchetes envolvendo o número da porta.

ACERTE O TOM

A instrução **nop**, com o código hex 0 0, significa "sem operação" — **no operation**. Ela não faz absolutamente nada, mas, como leva um certo tempo para ser executada — aproximadamente um microssegundo, ou seja, um milionésimo de segundo —, atrasa o microprocessador na realização do laço. Por essa razão, é usada duas vezes neste programa. A velocidade com que o microprocessador executa o laço controla a freqüência de vibração do cone do alto-falante e, conseqüentemente, a tonalidade do som produzido.

Como você pode observar, os **nop** não são executados apenas duas vezes. A instrução **djnz** (decrementa e salta se não for zero) promove várias repetições da pausa.

A instrução **djnz** opera com base no valor do registro B. Assim, na primeira execução do laço, ela decrementa o valor de B, que passa de 0 para 255. Em seguida, o laço é executado mais 255 vezes, até que B valha 0 novamente.

Uma vez concluído o laço, o valor no topo da pilha é recuperado com **pop** e colocado no par BC. Com a recuperação do valor original de B, **djnz** o decrementa e o microprocessador entra mais uma vez no laço. Esse processo leva o valor de BC de volta à pilha. Assim, o contador que fica na pilha é decrementado cada vez que o microprocessador executa o laço, e o valor inicial de B, que determina o número de execuções, torna-se menor. À medida que o número de execuções da pausa diminui, a freqüência do som aumenta.

MOLDURA COM SETE CORES

Para especificar a cor da moldura utilizando o comando BASIC **BORDER**, atribuímos a ela um valor qualquer entre 0 e 7. Toda a moldura fica, então, com a cor correspondente.

A rotina seguinte usa código de máquina para criar uma moldura com sete cores. Poderíamos trabalhar com oito cores, mas não faria sentido ter na moldura uma tonalidade idêntica à da tela principal.

```
 10 REM org 64600
 20 REM redo halt
 30 REM xor a
 40 REM loop out 254,a
 50 REM ld b,205
 60 REM pause ld e,2
 70 REM inner dec e
 80 REM jr nz,inner
 90 REM djnz pause
100 REM ld d,a
110 REM ld a,$7F
120 REM in a,254
130 REM rra
140 REM rts nc
150 REM ld a,d
160 REM inc a
170 REM cp 7
180 REM jr nz,loop
190 REM jr redo
```

COMO SINCRONIZAR A TELA

A instrução **halt** aguarda a ocorrência de uma interrupção para, depois, continuar o programa. No Spectrum, a interrupção ocorre a cada varredura da tela. Assim, **halt** inicia a rotina quando a varredura começa no alto da tela, sincronizando a posição da moldura com a borda superior da tela de TV.

A instrução **xor a** executa um "ou exclusivo" entre o acumulador e ele mesmo, o que constitui uma forma rápida de zerar o acumulador. O 0 é enviado pela porta 254. Como zero, em linguagem de máquina e em BASIC, corres-

Quantas portas existem?

Teoricamente, existem 64K portas — este é o maior número que pode ser endereçado por um par de registros de oito bits. Na prática, porém, utilizam-se apenas umas poucas portas. Quando seu Spectrum está na configuração original, somente a porta 254 é usada.

O comando **in** permite empregar diferentes parâmetros para dirigir a porta para as várias regiões do teclado. Neste artigo, recorremos a bits distintos, da mesma porta, para controlar periféricos diversos.

Se seu Spectrum for conectado a periféricos não usuais, talvez você tenha a possibilidade de aproveitar outras portas. Conhecendo razoavelmente eletrônica, poderá, por exemplo, ligar seu micro a um sistema de aquecimento ou a um sintetizador, utilizando portas diferentes.

ZAP!

ponde a preto, a porçao superior da tela adquire esta cor.

AJUSTE A PAUSA

Nesta rotina o tempo é essencial. O comprimento da pausa determina a largura das bandas coloridas da moldura. Controla-se a pausa por meio de dois laços: o interno, que acerta o tempo grosseiramente; e o externo, que o sintoniza de modo mais refinado.

O laço interno é executado com base no registro E, que recebe o valor 2 através de **ld e,2**, sendo decrementado por **dec e**. A instrução **jrnz,inner** realiza um salto relativo em direção ao rótulo **inner**, se o resultado não for zero. Assim, a cada execução do laço externo, o laço interno é executado duas vezes.

O laço externo repete-se 205 vezes com o registro B. **ld b,205** coloca este valor no registro B, assim como o **djnz** que o decrementa, saltando enquanto o resultado não for zero. Nenhuma instrução deste tipo opera com o registro E; por causa disso, a subtração e o salto precisam ser feitos em duas instruções separadas.

Se alterarmos tais valores, veremos que o valor colocado em E modifica bastante a largura das bandas, enquanto o valor colocado no registro B tem um efeito bem menor.

A OCORRÊNCIA DE UM BREAK

Certamente, em algum momento, desejaremos sair desse laço. Para fazê-lo sem desligar seu micro, programe uma rotina de saída.

A que apresentamos aqui verifica se a tecla <**BREAK**> foi pressionada, por meio da instrução **in**.

Inicialmente, porém, **ld d,a** coloca o valor do acumulador no registro D. Por um instante o acumulador será usado para outras coisas, e o registro D ficará desocupado, podendo servir de registro temporário.

Em seguida, o acumulador é carregado com o valor hexadecimal 7F. Esse número especifica o canto inferior esquerdo do teclado. A instrução **in a,254** recebe o padrão de bits correspondente ao estado atual dessa parte do teclado, colocando-o no acumulador.

O bit zero representa o estado da tecla <**BREAK**>. Se esta não estiver sendo pressionada, ele valerá 1; se estiver, seu valor será igual a 0.

Para verificar isso, execute uma rotação, jogando o bit zero — que fica no extremo direito do byte — para o carry bit e checando, depois, o sinalizador carry. A instrução **rra** promove a rotação e **rts nc** retorna ao BASIC quando não há um "vai um" — carry igual a zero ou "not carry". Assim, quando pressionar <**BREAK**> e o bit zero passar a valer zero, **rts** provocará o fim da rotina; caso contrário, esta continuará funcionando.

Lembre-se de que um retorno condicional tem o mnemônico *rts* apenas em nosso Assembler. Outros Assemblers utilizam a forma **ret**.

COMPLETE O CÍRCULO

Agora que temos uma saída da rotina, o valor do acumulador é recuperado por **ld a,d**, já que estava temporariamente em D.

A é incrementado para especificar a cor seguinte. A instrução **cmp 7** compara o resultado com 7. Este é o número correspondente ao branco, cor da tela principal. Se o número no acumulador não é sete, **jrnz,loop** volta para enviar a cor da próxima banda. Quando este laço tiver contado de 0 a 7, o microprocessador vai à instrução **jr redo**, que retorna ao início do programa, à espera de uma nova varredura da tela.

Você pode estar estranhando o fato de não produzir sons por acidente ao mudar a cor da moldura. A rotina não afeta o bit que controla o alto-falante. O maior número que enviamos pela porta 254 foi 7, quando precisaríamos de, no mínimo, 16 para movimentar o alto-falante.

EFEITOS SONOROS

Veremos agora como criar um efeito sonoro mais complexo, com o comando **out**. A rotina apresentada a seguir produz um som de disparo laser, usando a combinação de um som cuja freqüência é crescente com um som de freqüência decrescente.

```
 10 REM org 64600
 20 REM ld a,(23624)
 30 REM rrca
 40 REM rrca
 50 REM rrca
 60 REM ld b,0
 70 REM loop push bc
 80 REM xor $10
 90 REM out 254,a
100 REM push af
110 REM xor a
120 REM sub b
130 REM ld b,a
140 REM pop af
150 REM pausea nop
160 REM djnz pausea
170 REM xor $10
180 REM out 254,a
190 REM pop bc
200 REM push bc
210 REM pauseb nop
220 REM djnz pauseb
230 REM pop bc
240 REM djnz loop
250 REM ret
```

As quatro primeiras instruções são exatamente iguais às do outro programa de som. Têm a função de preservar a moldura.

As duas seguintes — que estabelecem os valores iniciais de dois contadores, um no registro B e outro colocado na pilha a partir do registro B — também são iguais. O mesmo ocorre com as outras duas, que usam "ou exclusivo" entre os bits quatro e dezesseis e mandam o resultado pela porta 254. Desta vez, porém, **xor** é seguido de $10 — 16, em hex. Isto produz o som.

O **push** guarda o conteúdo do acumulador na pilha, junto com o registro F. Os registros só podem ser guardados em pares. A instrução **xor a** limpa o acumulador e **sub b** subtrai B de 0 — conteúdo do acumulador. Na prática, porém, inverte-se o conteúdo de B. Quando B é 255, obtemos 1 como resultado da subtração; quando B é 1, obtemos como resultado 255.

O **ld b,a** coloca o resultado da subtração de volta em B. O acumulador que estava na pilha é novamente recuperado, retornando ao acumulador.

Temos, então, um laço de pausa cuja duração depende do conteúdo de B — lembre-se de que **djnz** decrementa o registro B e verifica se o resultado é zero. O valor do bit quatro é invertido e enviado ao alto-falante.

O **pop** recupera da pilha o valor inicial de B. Como este valor será utilizado mais adiante, é guardado novamente na pilha, depois de ter sido copiado em B. Agora ele se encontra, portanto, na pilha e em B.

A próxima pausa depende desse valor inicial de B, que mais uma vez é recuperado da pilha. O mesmo ocorre sempre que se executa uma instrução **djnz**, para restaurar o valor de B. Quando o processador sai de um laço **djnz**, o valor de B tem que ser zero.

O valor inicial de B é, então, diminuído. Não sendo zero, o processador retorna ao início do laço e executa-o novamente, com um valor menor que o de B. Na passagem 256, quando B se torna 0, o processador passa à instrução **ret** e retorna ao BASIC.

Você verá que o laço **pausea** é executado 256 vezes na primeira passagem, uma vez na segunda e uma vez mais a cada nova passagem. O laço **pauseb**, por outro lado, é feito 256 vezes na primeira passagem e uma vez a menos a cada nova volta.

ADIÇÃO DE SONS NATURAIS

Os sons que os computadores produzem parecem artificiais porque são muito puros. Instrumentos musicais e outros aparelhos que produzem sons tendem a ser bastante irregulares na maneira de produzi-los. Mas é justamente esta característica de acaso que lhes assegura efeitos tão agradáveis. Um instrumento não toca apenas o som fundamental, mas também os harmônicos.

Existe uma maneira de introduzir um elemento de acaso nos sons produzidos pelo Spectrum. A RAM, entre 16384 e 32767, fica em circuitos integrados que são geralmente interrompidos por um dispositivo que cuida da tela de TV e executa ainda outras tarefas.

Normalmente, essas interrupções são quase imperceptíveis, de tão curtas — em geral duram uns poucos microssegundos. Mas a relação entre os sons e o tempo é tão estreita que o ouvido pode captar variações mínimas. Assim, colocando o programa gerador de sons nessa área da memória, eles soarão bem mais naturais — e terão uma duração um pouco maior.

No nosso caso, a experiência não poderá ser feita, pois nosso Assembler ocupa essa área da memória. Contudo, se não estiver usando uma impressora, tente montar o programa em seu buffer, caso ele caiba ali. O buffer vai de 23296 a 23532 — assim, use 23296 como origem. Não é preciso proteger esse espaço, pois o buffer está a salvo de ser apagado pelo BASIC.

MICRO DICAS

FAÇA MÚSICA NO SPECTRUM

O código **out** permite a produção de música em seu micro. Mas esteja preparado: não é fácil executar os ajustes necessários para obter a freqüência correta.

O uso de uma rotina com dois parâmetros — relacionados à freqüência e à duração — simplifica bastante a tarefa. E esta rotina existe na ROM. Chamada de rotina BEEP, começa no endereço 03f8. Existe ainda uma rotina adicional chamada BEEPER, que utiliza o valor em HL, para controlar a freqüência, e o valor em DE, para controlar a duração.

O valor que se deve usar em HL é dado por 437500/f-301125, onde f é a freqüência da nota. Multiplicando a freqüência pela duração desejada, obtém-se o valor a ser colocado em DE. Com este método, não é preciso observar o limite usual de 10 segundos para a duração da nota.

Se você chamar a rotina BEEP diretamente, deverá colocar na pilha os mesmos valores de duração e freqüência que são empregados num comando **SOUND** comum.

LINHA	FABRICANTE	MODELO
Apple II +	Appletronica	Thor 2010
Apple II +	CCE	MC-4000 Exato
Apple II +	CPA	Absolutus
Apple II +	CPA	Polaris
Apple II +	Digitus	DGT-AP
Apple II +	Dismac	D-8100
Apple II +	ENIAC	ENIAC II
Apple II +	Franklin	Franklin
Apple II +	Houston	Houston AP
Apple II +	Magnex	DM II
Apple II +	Maxitronica	MX-2001
Apple II +	Maxitronica	MX-48
Apple II +	Maxitronica	MX-64
Apple II +	Maxitronica	Maxitronic I
Apple II +	Microcraft	Craft II Plus
Apple II +	Milmar	Apple II Plus
Apple II +	Milmar	Apple Master
Apple II +	Milmar	Apple Senior
Apple II +	Omega	MC-400
Apple II +	Polymax	Maxxi
Apple II +	Polymax	Poly Plus
Apple II +	Spectrum	Microengenho I
Apple II +	Spectrum	Spectrum ed
Apple II +	Suporte	Venus II
Apple II +	Sycomig	SIC I
Apple II +	Unitron	AP II
Apple II +	Victor do Brasil	Elppa II Plus
Apple II +	Victor do Brasil	Elppa Jr.
Apple IIe	Microcraft	Craft IIe
Apple IIe	Microdigital	TK-3000 IIe
Apple IIe	Spectrum	Microengenho II
MSX	Gradiente	Expert GPC-1
MSX	Sharp	Hotbit HB-8000
Sinclair Spectrum	Microdigital	TK-90X
Sinclair Spectrum	Timex	Timex 2000
Sinclair ZX-81	Apply	Apply 300
Sinclair ZX-81	Engebras	AS-1000
Sinclair ZX-81	Filcres	NEZ-8000
Sinclair ZX-81	Microdigital	TK-82C
Sinclair ZX-81	Microdigital	TK-83
Sinclair ZX-81	Microdigital	TK-85
Sinclair ZX-81	Prologica	CP-200
Sinclair ZX-81	Ritas	Ringo R-470
Sinclair ZX-81	Timex	Timex 1000
Sinclair ZX-81	Timex	Timex 1500
TRS-80 Mod. I	Dismac	D-8000
TRS-80 Mod. I	Dismac	D-8001/2
TRS-80 Mod. I	LNW	LNW-80
TRS-80 Mod. I	Video Genie	Video Genie I
TRS-80 Mod.III	Digitus	DGT-100
TRS-80 Mod.III	Digitus	DGT-1000
TRS-80 Mod.III	Kemitron	Naja 800
TRS-80 Mod.III	Prologica	CP-300
TRS-80 Mod.III	Prologica	CP-500
TRS-80 Mod.III	Sysdata	Sysdata III
TRS-80 Mod.III	Sysdata	Sysdata Jr.
TRS-80 Mod.IV	Multix	MX-Compacto
TRS-80 Mod.IV	Sysdata	Sysdata IV
TRS-Color	Codimex	CS-6508
TRS-Color	Dynacom	MX-1600
TRS-Color	LZ	Color 64
TRS-Color	Microdigital	TKS-800
TRS-Color	Prologica	CP-400

FABRICANTE	MODELO	PAÍS	LINHA
Appletronica	Thor 2010	Brasil	Apple II +
Apply	Apply 300	Brasil	Sinclair ZX-81
CCE	MC-4000 Exato	Brasil	Apple II +
CPA	Absolutus	Brasil	Apple II +
CPA	Polaris	Brasil	Apple II +
Codimex	CS-6508	Brasil	TRS-Color
Digitus	DGT-100	Brasil	TRS-80 Mod.III
Digitus	DGT-1000	Brasil	TRS-80 Mod.III
Digitus	DGT-AP	Brasil	Apple II +
Dismac	D-8000	Brasil	TRS-80 Mod. I
Dismac	D-8001/2	Brasil	TRS-80 Mod. I
Dismac	D-8100	Brasil	Apple II +
Dynacom	MX-1600	Brasil	TRS-Color
ENIAC	ENIAC II	Brasil	Apple II +
Engebras	AS-1000	Brasil	Sinclair ZX-81
Filcres	NEZ-8000	Brasil	Sinclair ZX-81
Franklin	Franklin	USA	Apple II +
Gradiente	Expert GPC1	Brasil	MSX
Houston	Houston AP	Brasil	Apple II +
Kemitron	Naja 800	Brasil	TRS-80 Mod.III
LNW	LNW-80	USA	TRS-80 Mod. I
LZ	Color 64	Brasil	TRS-Color
Magnex	DM II	Brasil	Apple II +
Maxitronica	MX-2001	Brasil	Apple II +
Maxitronica	MX-48	Brasil	Apple II +
Maxitronica	MX-64	Brasil	Apple II +
Maxitronica	Maxitronic I	Brasil	Apple II +
Microcraft	Craft II Plus	Brasil	Apple II +
Microcraft	Craft IIe	Brasil	Apple IIe
Microdigital	TK-3000 IIe	Brasil	Apple IIe
Microdigital	TK-82C	Brasil	Sinclair ZX-81
Microdigital	TK-83	Brasil	Sinclair ZX-81
Microdigital	TK-85	Brasil	Sinclair ZX-81
Microdigital	TK-90X	Brasil	Sinclair Spectrum
Microdigital	TKS-800	Brasil	TRS-Color
Milmar	Apple II Plus	Brasil	Apple II +
Milmar	Apple Master	Brasil	Apple II +
Milmar	Apple Senior	Brasil	Apple II +
Multix	MX-Compacto	Brasil	TRS-80 Mod.IV
Omega	MC-400	Brasil	Apple II +
Polymax	Maxxi	Brasil	Apple II +
Polymax	Poly Plus	Brasil	Apple II +
Prologica	CP-200	Brasil	Sinclair ZX-81
Prologica	CP-300	Brasil	TRS-80 Mod.III
Prologica	CP-400	Brasil	TRS-Color
Prologica	CP-500	Brasil	TRS-80 Mod.III
Ritas	Ringo R-470	Brasil	Sinclair ZX-81
Sharp	Hotbit HB-8000	Brasil	MSX
Spectrum	Microengenho I	Brasil	Apple II +
Spectrum	Microengenho II	Brasil	Apple IIe
Spectrum	Spectrum ed	Brasil	Apple II +
Suporte	Venus II	Brasil	Apple II +
Sycomig	SIC I	Brasil	Apple II +
Sysdata	Sysdata III	Brasil	TRS-80 Mod.III
Sysdata	Sysdata IV	Brasil	TRS-80 Mod.IV
Sysdata	Sysdata Jr.	Brasil	TRS-80 Mod.III
Timex	Timex 1000	USA	Sinclair ZX-81
Timex	Timex 1500	USA	Sinclair ZX-81
Timex	Timex 2000	USA	Sinclair Spectrum
Unitron	AP II	Brasil	Apple II +
Victor do Brasil	Elppa II Plus	Brasil	Apple II +
Victor do Brasil	Elppa Jr.	Brasil	Apple II +
Video Genie	Video Genie I	USA	TRS-80 Mod. I

INPUT foi especialmente projetado para microcomputadores compatíveis com as sete principais linhas existentes no mercado.
Os blocos de textos e listagens de programas aplicados apenas a determinadas linhas de micros podem ser identificados por meio dos seguintes símbolos:


Sinclair ZX-81


TRS-80


TK-2000


MSX


Spectrum


TRS-Color


Apple II

Quando o emblema for seguido de uma faixa, então tanto o texto como os programas que se seguem passam a ser específicos para a linha indicada.

NO PRÓXIMO NÚMERO

APLICAÇÕES

A dificuldade de preparar um texto é coisa do passado. Resolva seus velhos problemas com um programa editor.

CÓDIGO DE MÁQUINA

Se você é usuário do Spectrum ou do TRS-Color, veja como funciona o gerador gráfico de nossos programas de animação.

PROGRAMAÇÃO BASIC

Complete o cenário montado com UDG. E aproveite para examinar algumas idéias a respeito de animação e gravação.

PERIFÉRICOS

Seu computador pode se comunicar com outros. O modem é um bom elo de ligação.